КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

НА ТЕМУ:

«Інтелектуальна система для підтримки прийняття рішень при оцінюванні відповідей студентів на відкриті питання контрольних заходів»

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітньо-наукова програма «Технології штучного інтелекту»

Виконав:
студент 2 курсу магістратури, групи ТШІ-21
Черненко А.Є.
(ПІБ)

Науковий керівник: Тмєнова Н.П.
(ПІБ)
к. ф-м. н., доцент
(науковий ступінь, вчене звання)

Засвідчую, що в цій кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань

Студент
підпис

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол № 12 від « 11 » травня 2023 р.

Зав. кафедри _____ доц. Іларіонов О.Є.
підпис

Київ - 2023

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, 3 розділів, висновків, списку використаної літератури із 44 джерел та 9 додатків. Загальний обсяг роботи 143 сторінки. Робота містить 2 таблиці та 21 рисунок.

Актуальність теми. Процес перевірки контрольної роботи вимагає від викладача постійної зосередженості, максимальної об’єктивності оцінювання та швидкої перевірки. Досягти комбінації цих факторів важко для будь-якої людини, а якщо вона їх досягає, то це дуже її навантажує. Тож для оптимізації перевірки робіт студентів та визначення оцінок цих робіт доцільним є використовувати спеціальне програмне забезпечення і потужності комп’ютера. Оскільки таких специфічних програмних застосунків немає, тому ця робота є актуальною.

Мета роботи: створення за допомогою підходів NLP та нейромережевих технологій інтелектуальної системи для підтримки прийняття рішень при оцінюванні відповідей студентів на відкриті питання контрольних заходів та аналіз її ефективності.

Об’єкт дослідження – процес порівняння текстів на їх лексичну, синтаксичну та семантичну подібність за допомогою інструментів NLP та нейромережевих технологій.

Предмет дослідження – система для аналізу та порівняння текстів за допомогою засобів NLP та нейромережевих технологій.

Результати роботи. Програмно реалізована інтелектуальна система для підтримки прийняття рішень при оцінюванні відповідей студентів на відкриті питання контрольних заходів

Апробація роботи. Система пройшла тестування

Ключові слова. контрольні заходи, студент, викладач, оцінювання, обробка природномовної інформації, n-грама, нейронна мережа.

ABSTRACT

The qualification work consists of an introduction, 3 chapters, conclusions, a list of used literature from 44 sources and 9 appendices. The total volume of work is 143 pages. The work contains 2 tables and 21 figures.

Theme`s relevance. The process of checking the test work requires constant concentration from the teacher, maximum objectivity of the assessment and quick check. Achieving a combination of these factors is difficult for any person, and if person achieves them, it is very stressful for this human being. Therefore, it is reasonable to use special software and computer power to optimize the examination of students' works and determine the grades of these works. But there are no such specific software applications, so this work is relevant.

The purpose of the work: to create, using NLP approaches and neural network technologies, an intelligent system for decision support when evaluating students' answers to open questions of control measures and analyzing its effectiveness.

The object of the research is the process of comparing texts for their lexical, syntactic and semantic similarity using NLP tools and neural network technologies.

The subject of the research is a system for analyzing and comparing texts using NLP tools and neural network technologies.

Work results. Software-implemented intelligent system to support decision-making when evaluating students' answers to open questions of control measures

Approbation of work. The system has been tested

Keywords: control measures, student, teacher, assessment, natural language processing, n-gram, neural network.

ЗМІСТ

ВСТУП

В XXI сторіччі цифрові технології ввійшли майже в кожну сферу життєдіяльності людини. Зараз важко уявити свої щоденні активності без використання різних гаджетів, що значно спрощують наше життя. Вони мають дуже різноманітний функціонал, котрий щодня розширюється, надаючи нові можливості зробити наше життя кращим та легшим. Однією з сфер життєдіяльності, яка активно розвиває цифрову складову, є сфера освіти, особливо вищої освіти.

Раніше освіта ґрунтувалась на засвоєнні матеріалу наживо. Необхідними були відвідування уроків в школі, або пар в університеті. Це створювало певні незручності у випадках форс-мажорних ситуацій у студентів і викладачів. Зі збільшенням залученості цифрових технологій у цифровий процес такі ситуації стали менше впливати на навчальний процес, оскільки тепер можна розміщувати навчальні матеріали онлайн або надсилати роботи для попередньої перевірки викладачем. Це дає певну гнучкість та зручність.

Важливішим за зручність є фактор необхідності. Останні декілька років були дуже показовими: спочатку епідемія COVID-19, потім війна. Навчальні заклади вимушені працювати дистанційно, щоб не наражати учнів/студентів на небезпеку. І в цьому випадку гарний рівень організації дистанційного навчання є критично важливим. Це стосується і організації контрольних заходів.

В загальному випадку контрольні заходи проводяться в 3 видах: тести, практичні завдання та відкриті запитання. І кожен з них необхідно оцінювати. Перший вид цього не потребує, оскільки більшість сервісів, що надають можливість створення та проходження тестів, мають і автоматичну перевірку відповідей. Другий вид є дещо складним в реалізації, оскільки практичні роботи можуть варіюватися для різних спеціальностей і потребують конкретно

створений застосунок. Тому ми зупинимось на третьому виді. Для його реалізації необхідне використання методів обробки природномовної інформації (NLP).

Методи NLP доволі стрімко розвиваються та широко застосовуються в повсякденному житті. Це і машинний переклад, і системи допомоги в написанні повідомлень, і голосове управління, голосові помічники та генерація мови (перекладеної), і анотування текстів, і контекстна реклама, і багато іншого. Для того, щоб виконувати такий великий спектр робіт, NLP агрегує в собі велику кількість методів, підходів та інших компонентів. Тому основним завданням даної роботи буде більш глибоке знайомство з поняттям NLP та його підходами і їх застосування в реалізації інтелектуальної системи для підтримки прийняття рішень при оцінюванні відповідей студентів на відкриті питання контрольних заходів.

1 Аналітичний огляд проблеми оцінювання відповідей студентів на відкриті питання та постановка завдання для розробки системи їх автоматичної перевірки

1.1 Актуальність проблеми

Як було сказано у вступі роботи, сфера освіти проходить свою трансформацію та переміщення у цифровий простір. Це стосується і оцінювання контрольних робіт студентів. І на поточний момент оцінювання відповідей на контрольні питання виконується викладачами вручну. Це має певні недоліки.

Першим недоліком є наявність у викладача ряду ознак або суб'єктивних рис, які притаманні людині, і які можуть вплинути на раціональне оцінювання роботи студента. Наприклад, викладачу можуть бути властиві такі характеристики як хвороба, поганий настрій або інші особливості, що можуть негативно відобразитись на процесі оцінювання і зробити його необ'єктивним.

Другим недоліком є власне особисте залучення викладача до процесу оцінювання, що може призвести до ситуації обвинувачення викладача в упередженому відношенні до студента.

Третім недоліком є витрата часу. Викладач витрачає свій час та сили на перевірку робіт, а студент повинен отримати результат через деякий час після здачі роботи, що також може бути не зручним.

Щоб подібних ситуацій не виникало, краще, щоб перевірку відповідей на відкриті питання виконувала автоматична система, що зробить процес оцінювання об'єктивним та швидким [1].

На поточний момент достатньо ефективних систем не було знайдено або впроваджено для українських закладів вищої освіти на загальних основах, тому я вважаю, що робота на дану тему є доволі актуальною і має великий потенціал в майбутньому при професійній розробці.

1.2 Аналіз предметної області, існуючих бізнес-процесів

Областю застосування розроблюваної системи є оцінювання відповідей студентів на відкриті питання контрольних заходів в рамках навчального процесу в вищому навчальному закладі.

Вищий навчальний заклад надає освітні послуги студентам. Ці послуги чітко регламентуються, і важливою частиною регламенту є перевірка робіт студентів на контрольних заходах. Викладач приймає написану роботу студента та виконує перевірку на основі власних знань та суджень. Після ретельної перевірки, викладач робить певні висновки і оцінює роботу студента за набором критеріїв (власних або загальних). Після цього студент повідомляється про оцінку і отримує право її оскаржити за допомогою особистої бесіди або перескладання з комісією.

1.3 Опис профілів зацікавлених сторін

Опис профілів зацікавлених сторін (стейкхолдерів) наведено у таблиці 1.3.

Таблиця 1.3 Профілі зацікавлених сторін.

Зацікавлена сторона	Здобута вигода	Очікування (неявні потреби)	Основні інтереси	Обмеження
Розробник	Прибуток Підвищення іміджу. Нові клієнти. Набуття досвіду.	Заняття ніші на ринку ПЗ. Можливість подальшого покращення.	Функціонал повинен виконувати поставлені задачі.	Обмеженість фінансами. Обмеженість часом розробки.

Кінець таблиці 1.3

Зацікавлена сторона	Здобута вигода	Очікування (неявні потреби)	Основні інтереси	Обмеження
			Ефективність рішень більша за аналоги. Максимізація прибутку.	
Викладач	Зменшення об’єму праці. Зменшення суб’єктив-ності в оцінюванні. Прискорення процесу оцінювання.	Зменшення витрат часу на 50% при збереженні якості оцінювання.	Ефективність оцінювання на прийнятному рівні. Максимальний функціонал. Обґрунтована ціна. Простота в експлуатації. Постійна працездатність.	Обмеженість власними знаннями.
Студент	Прискорення процесу оцінювання.	Збільшення кількості особистого часу.	Неупередже-ність оцінювання. Швидкий результат.	Немає.

1.4 Існуючі підходи. Сутність та підходи NLP

1.4.1 Сутність NLP

Обробка природної мови (Natural language processing) — це аналіз лінгвістичних даних, найчастіше у формі текстових даних, таких як документи чи публікації, за допомогою обчислювальних методів. Метою обробки природної мови є, як правило, побудова представлення тексту, яке додає структуру неструктурованій природній мові, використовуючи переваги лінгвістичних ідей. Ця структура може бути синтаксичною за своєю природою – фіксувати граматичні зв’язки між складовими тексту – або більш семантичною – фіксувати значення, передане текстом [2].

Обробку природної мови, в основному, можна поділити на дві частини, тобто розуміння природної мови та генерацію природної мови, що розвиває завдання розуміння та створення тексту [3].

Рівні, на яких оперує NLP, можна розділити наступним чином:

1. Фонологія - це розділ лінгвістики, який вивчає систематичне розташування звуків. Фонологія займається функцією, поведінкою та організацією звуків як мовних елементів. Фонологія включає семантичне використання звуку для кодування значення будь-якої людської мови.

2. Різні частини слова являють собою найменші одиниці значення, відомі як морфеми. Морфологія, яка складається з природи слів, ініційована морфемами. Прикладом морфеми може бути те, що слово може бути морфологічно розкладене на три окремі морфеми: префікс, корінь та суфікс. Інтерпретація морфеми залишається однаковою для всіх слів, просто щоб зрозуміти значення, люди можуть розбити будь-яке невідоме слово на морфеми [3]. При обробці слів часто бажано нормалізувати використану варіацію слова до його основної форми, щоб правильно зв’язати різні випадки того ж самого

терміну. Це називається морфологічною нормалізацією і часто досягається в практичних застосуваннях NLP за допомогою інструментів, які видаляють відмінювані закінчення слів [2].

3. У лексиці люди, а також системи NLP, інтерпретують значення окремих слів. Різні типи обробки сприяють розумінню на рівні слова, перший із них – тег частини мови для кожного слова. Під час цієї обробки словам, які можуть виступати як декілька частин мови, призначається тег найбільш імовірної частини мови на основі контексту, у якому вони зустрічаються. На лексичному рівні семантичні уявлення можна замінити словами, які мають одне значення. У системі NLP характер репрезентації змінюється відповідно до розгорнутої семантичної теорії.

4. Синтаксичний рівень необхідний для ретельного вивчення слів у реченні, щоб розкрити граматичну структуру речення. На цьому рівні потрібні і граматика, і синтаксичний аналізатор. Результатом цього рівня обробки є представлення речення, яке розкриває зв’язки структурної залежності між словами [3]. Це досягається за допомогою інструментів, які виконують тегування слів відповідною частиною мови (POS) [2].

5. Семантична обробка визначає можливі значення речення, орієнтуючись на взаємодію між значеннями на рівні слів у реченні. Цей рівень обробки може включати семантичне усунення неоднозначності слів із кількома значеннями; схожим чином на те, як використовується синтаксичне усунення неоднозначності слів, які можуть виконувати роль кількох частин мови.

6. У той час як синтаксис і семантика мають проблеми з одиницями довжини речення, рівень дискурсу NLP має проблеми з одиницями тексту, довшими за речення, тобто він не інтерпретує тексти з кількома реченнями лише як послідовні речення, кожен з яких можна роз’яснити окремо. Дискурс можна поділити на два види: розв’язання анафори – це заміна таких слів, як займенники,

та розпізнавання структури дискурсу/тексту, що впливає на функції речень у тексті, що, у свою чергу, додає значущості представлення тексту.

7. Прагматичний рівень використовує суть тексту, щоб зрозуміти мету та пояснити, як додаткове значення закладається в тексти без буквального кодування в них. Для цього необхідно багато знань про світ, включаючи розуміння намірів, планів і цілей [3].

Для того, щоб ефективно оперувати на всіх цих рівнях, системи NLP використовують нормалізацію та приведення речень до виду, який здатна обробити машина. Одними із засобів для цього є токенізація та демаркація речень.

Обробка природної мови значною мірою базується на словах, оскільки зазвичай вважається, що слова несуть значення тексту. Тому важливо як етап попередньої обробки будь-якого подальшого аналізу розмежувати окремі лексеми слів, які складають текст. Цей процес називається токенізація. Хоча простий підхід полягає в тому, щоб розділити текст на будь-які пробіли або знаки пунктуації, у текстах потрібно бути обережним, щоб належним чином обробляти розділові знаки, які мають особливе значення в певних контекстах, наприклад скорочення [2].

Переваги NLP:

- Менші витрати: використання програми, оснащеної засобами NLP обходиться дешевше, ніж використання можливостей людини. Для виконання завдань людині може знадобитися в два-три рази більше часу, ніж машині.
- Швидший час відповіді служби підтримки клієнтів: зазвичай, коли використовується NLP, час відповіді чат-бота дуже швидкий [4]. Також на це не витрачається час та ресурси людини-оператора, який може краще розподілити свій час і реагувати тільки на важливі та складні ситуації.

- Автоматизація певних видів робіт: це дозволяє не тільки автоматично виконувати такі роботи, як написання статей, анотування та ін., але й значно пришвидшити та полегшити роботу.
- Обробка великого обсягу інформації: дозволяє проводити текстовий аналіз усіх типів документів, таких як електронні листи, соціальні мережі, онлайн-огляди, внутрішні системи тощо. Крім того, може обробити великі обсяги даних лише за кілька секунд, або, якщо дані занадто великі, він може зробити це за одну-дві хвилини [5].
- Акцент на головному: за допомогою засобів NLP можна видаляти непотрібну інформацію, а також знаходити потрібну доволі швидко, і використовувати певні методи, що здатні допомагати при різному роду аналізі.

Недоліки NLP:

- Навчання може зайняти час: якщо необхідно розробити модель із новим набором даних без використання попередньо навченої моделі, для досягнення високої продуктивності може знадобитися велика кількість часу залежно від обсягу даних.
- Використання NLP не на 100% надійно: один із недоліків машинного навчання полягає в тому, що воно ніколи не буває на 100% надійним. Існує ймовірність помилок у його прогнозі та результатах [4].
- Необхідність експертного втручання для створення коректного застосунку: в залежності від складності мови та текстів, що аналізуються, може знадобитись перевірка експертів для формування коректного застосунку.
- Недостатність досліджень: для поширених мов NLP надає гарні засоби для обробки та аналізу, але існує дуже багато мов і діалектів, що не поширені, а тому не представлені в достатній мірі або взагалі засобами NLP [5].
- Вживання в протилежному сенсі: деякий гумор і сарказм можуть мати на увазі протилежне до того, що слова в виразі означають, але моделі

машинного навчання недостатньо розвинені, щоб це опрацювати. Машина буде обробляти такі записи більш буквально.

1.4.2 Попередня обробка тексту

Попередня обробка тексту - це процес підготовки текстових даних, щоб машини могли використовувати їх для виконання таких завдань, як аналіз, прогнозування тощо [6].

Метою підготовки даних є створення «чистого тексту», який машини можуть аналізувати без помилок. Чистий текст - це людська мова, перебудована у формат, зрозумілий машинним моделям.

Нормалізація тексту - це процес стандартизації тексту, щоб за допомогою NLP комп'ютерні моделі могли краще розуміти вхідний текст, з кінцевою метою більш ефективної роботи аналізу.

Зокрема, нормалізація тексту може означати стандартизацію великих літер, щоб моделі машин не групували слова з великої літери як відмінні від їхніх аналогів у нижньому регістрі. Це називається нормалізацією регістру.

Пунктуація, Emoji, URL-адреси та @ заплутують моделі штучного інтелекту, оскільки вони є унікальними підписами, які або в кінцевому підсумку без користі перекладаються в Юнікод (усміхнене обличчя стає \u200c або подібним), або є унікальними (у випадку @ і гіперпосилань).

Пунктуація також створює шум і перешкоджає розумінню тексту засобами NLP, оскільки вона пов'язана з тоном конкретного речення, а не обов'язково зі словом, до якого воно прикріплене [7]. Тому необхідним є правильно обробити такі випадки і видалити відповідні записи в тексті, якщо вони не є важливими.

1.4.2.1 Стоп-слова. Видалення стоп-слів

Слова, які зазвичай відфільтровуються перед обробкою природної мови, називаються стоп-словами. Насправді це найпоширеніші слова в будь-якій мові (наприклад, артиклі, прийменники, займенники, сполучники тощо), і вони не додають багато інформації до тексту [6].

Стоп-слова доступні у великій кількості в будь-якій людській мові. Видаляючи ці слова, ми видаляємо інформацію низького рівня з нашого тексту, щоб приділити більше уваги важливій інформації. Одним словом, ми можемо сказати, що видалення таких слів не демонструє жодних негативних наслідків для моделі, яку ми тренуємо для нашого завдання [6].

Але слід сказати, що ми не завжди прибираємо стоп-слова. Видалення стоп-слів сильно залежить від завдання, яке ми виконуємо, і мети, яку ми хочемо досягти. Наприклад, якщо ми навчаємо модель, яка може виконувати завдання аналізу почуття, ми можемо не видаляти стоп-слова [6].

Є два принципових міркування щодо недоцільності видаляти стоп-слова для певної програми. Хоча стоп-слова позбавлені лексичного вмісту, під чим мається на увазі, що у них відсутні посилання на певну якість зовнішнього світу, який описується даним реченням, вони не зовсім позбавлені змісту.

У деяких випадках стоп-слова справді додають уточнення, і якщо програма чутлива до таких значень, стоп-слова не слід виключати.

Друге міркування полягає в тому, чи втрачається важлива інформація через їх видалення [8].

1.4.2.2 Токенізація

Токенізація — це процес поділу тексту на менші одиниці, які називаються лексемами (наприклад, слова). Це фундаментальний етап попередньої обробки

майже для всіх програм NLP: аналіз настроїв, відповіді на запитання, машинний переклад, пошук інформації тощо. Сучасні моделі NLP, такі як BERT, ГПТ-3 і XL-Net токенізують текст на підслівні одиниці. Будучи серединою між словами та символами, підслівні одиниці зберігають лінгвістичне значення (як морфеми), вирішуючи ситуації поза словниковим запасом навіть при відносно малооб'ємній лексиці [9].

Список токенів перетворюється на вхідні дані для додаткової обробки, включаючи розбір або видобуток тексту. Токенізація корисна як у лінгвістиці, так і в інформатиці, де вона є частиною лексичного аналізу.

Як правило, процес токенізації відбувається на рівні слова. Але іноді буває важко визначити, що мається на увазі під словом . Регулярно токенізатор використовує прості евристики, наприклад:

- пунктуація та пробіли можуть або не можуть бути включені до кінцевого списку токенів;
- усі суміжні рядки букв алфавіту є частиною одного токена; так само з числами;
- токени відокремлюються пробілами, такими як пробіл або розрив рядка, або знаками пунктуації.

Недоліком токенізації є те, що документ важко токенізувати без пробілів, спеціальних символів чи інших позначок.

Для токенізації документа доступно багато інструментів. Інструменти перераховані нижче:

- Токенізатор Nlpdotnet.
- Токенізатор Міла.
- NLTK Word Tokenize.
- TextBlob Word Tokenize.
- MBSP Word Tokenize.
- Шаблон Word Tokenize.

- Токенізація Word за допомогою Python NLTK [10].

Традиційний підхід до виконання завдання токенізації використовує створені вручну правила з регулярними виразами та/або кінцеві автомати. MtSeg, система, розроблена П. ді Крісто, є система для токенізації на основі правил, яку можна налаштувати для різних мов. Система була вдосконалена за допомогою відповідних ресурсів для кількох західноєвропейських мов. Другий підхід до токенізації базується на алгоритмах послідовного маркування, таких як приховані марковські моделі (HMM). В система розробленій Гіллом та співавторами токенізація та тегування частиною мови виконується за один крок за допомогою HMM алгоритм [11].

Тож можна навести декілька варіантів токенізації: попереднє токенізація тексту на слова (шляхом розділення на знаки пунктуації та пробіли) та токенізація кожного слова на фрагменти [9].

1.4.2.3 Стемінг

Було помічено, що у більшості випадків морфологічні варіанти слів мають подібні семантичні інтерпретації та можуть розглядатися як еквівалентні для цілей їх застосування. Оскільки значення однакове, але форма слова відрізняється, необхідно ототожнювати кожну словоформу з її основою. Для цього було розроблено різноманітні алгоритми стемінгу [12].

Стемінг визначається як процес, який створює варіанти кореня/основи слова. Простими словами, це скорочує базове слово до його кореня. Ми використовуємо ці корені, щоб скоротити пошук і нормалізувати речення для кращого розуміння [13].

Застосування стемінгу:

- інформаційно-пошукові системи;
- визначення словників домену в доменному аналізі.

Алгоритми стемінгу.

Різні алгоритми допомагають у виконанні стемінгу. Наведемо приклади деяких з алгоритмів стемінгу.

Найпростішим стемером був Truncate (n) stemmer, який скорочував слово на n-му символі, тобто зберігав n літер і видаляв решту. У цьому методі слова, коротші за n, зберігаються в тому вигляді, як є. Імовірність перетворення кореня збільшується, коли довжина слова мала.

Іншим простим підходом був S-stemmer – алгоритм, що об'єднує форми однини та множини англійських іменників. Цей алгоритм запропонувала Донна Харман. Алгоритм має правила видалення суфіксів у множині, щоб перетворити їх у форми однини [12].

Стемер Ловінса виконує пошук у таблиці з 294 закінченнями, 29 умовами та 35 правилами перетворення, які впорядковані за принципом найдовшого збігу. Основний ключ стемера Ловінса видаляє зі слова найдовший суфікс. Після видалення закінчення слово перекодується за допомогою іншої таблиці, яка вносить різні коригування, щоб перетворити ці основи на дійсні слова. Він завжди видаляє максимум один суфікс зі слова через його природу однопрохідного алгоритму.

Перевагами цього алгоритму є те, що він дуже швидкий і може обробляти видалення подвійних літер, а також обробляє багато неправильних форм множини.

Недоліки стемера Ловінса полягають у тому, що він вимагає багато часу та даних. Крім того, багато суфіксів недоступні в таблиці закінчень. Іноді він дуже ненадійний і часто не може скласти слова з основ або зіставити основи слів зі схожим значенням. Причина – технічний словник, який використовує автор [12].

Стемер Портера. Цей алгоритм визначення кореня використовується для видалення та заміни суфіксів англійських слів, щоб зробити слова простішими та ефективнішими [13]. Він заснований на ідеї, що суфікси в англійській мові

(приблизно 1200) здебільшого складаються з комбінації менших і простіших суфіксів. Він має п’ять кроків, і на кожному кроці застосовуються правила, доки одне з них не виконає умови. Якщо правило прийнято, суфікс видаляється відповідним чином і виконується наступний крок. Результуюча основа в кінці п'ятого кроку повертається [12]. NLTK має клас PorterStemmer, за допомогою якого ми можемо реалізувати алгоритм Porter Stemmer. Використовуючи цей клас, ми можемо перетворити задане текстове слово на результуючу основу, яка, у свою чергу, дає коротше слово з таким же кореневим значенням [13].

Стемер Paice/Husk або Ланкастерський стемер — це ітераційний алгоритм з однією таблицею, що містить близько 120 правил, індексованих останньою літерою суфікса. На кожній ітерації він намагається знайти відповідне правило за останнім символом слова. Кожне правило визначає видалення або заміну закінчення. Якщо такого правила немає, він припиняється. Він також закінчується, якщо слово починається з голосної і залишилося лише дві літери, або якщо слово починається з приголосної і залишилося лише три символи. В іншому випадку застосовується правило і процес повторюється [12]. Це дуже поширена техніка стемінга. NLTK має клас LancasterStemmer, за допомогою якого ми можемо реалізувати алгоритм Lancaster Stemming для виконання стемінга [13].

Перевагою є його проста форма та кожна ітерація, яка передбачає як видалення, так і заміну відповідно до застосованого правила.

Недоліком є те, що це дуже важкий алгоритм, і може виникнути надмірне стемінування [12].

Стемер Доусона є розширенням стемера Ловінса, за винятком того, що він охоплює набагато повніший список приблизно з 1200 суфіксів. Як і стемер Ловінса, це також однопрохідний стемер і, отже, досить швидкий. Суфікси зберігаються у зворотному порядку з індексами їх довжини та останньої літери.

Насправді вони організовані як набір розгалужених дерев символів для швидкого доступу.

Перевагою є те, що він охоплює більше суфіксів, ніж стемер Ловінса, і швидко виконується.

Недоліком є те, що він дуже складний і не має стандартної багаторазової реалізації [12].

Стемер на основі n-грам - це дуже цікавий метод, який не залежить від мови. Тут використовується підхід подібності рядків, щоб перетворити інфляцію слова в його основу. В даному випадку n-грама виступає в якості рядка із n, зазвичай суміжних, символів, виділених із частини безперервного тексту. Якщо бути точним, то n-грама в даному випадку - це набір із n послідовних символів, виділених із слова. Основна ідея цього підходу полягає в тому, що подібні слова матимуть велику частку спільних n-грам.

Цей стемер має перевагу в тому, що він не залежить від мови і тому дуже корисний у багатьох програмах.

Недолік полягає в тому, що для створення та зберігання n-грам та індексів потрібен значний обсяг пам’яті та сховища, а отже, це не дуже практичний підхід [12].

Алгоритм на основі регулярного виразу.

За допомогою алгоритму стемінування Regexp ми можемо створити власний стемер. NLTK має клас RegexpStemmer, за допомогою якого ми можемо реалізувати алгоритми Stemmer з регулярними виразами. Він приймає один вираз як вхідні дані та видаляє суфікс і префікс, які відповідають виразу.

Алгоритм створення сніжної кулі.

Це дуже корисний алгоритм стемінгу. NLTK має клас SnowballStemmer, за допомогою якого ми можемо реалізувати алгоритми Snowball Stemmer. Він підтримує 15 мов, не тільки англійську. Для використання цього класу ми

повинні побудувати підформу з назвою мови, яку ми використовуємо, а потім використати метод stem() для її успішного створення [13].

1.4.2.4 Лематизація

Це процес збирання відмінюваних частин слова таким чином, щоб їх можна було розпізнати як єдиний елемент, який називається лемою слова або його словниковою формою. Вона визначається як метод знаходження леми слова, яке є коренем, а не основою і заснована на передбачуваному значенні, яке намагається передати слово [13].

Лематизація має справу зі складним процесом спочатку розуміння контексту, потім визначення позиції слова в реченні, а потім, нарешті, пошуку «леми» [12].

Загальне правило щодо того, чи потрібно лематизувати, не дивне: якщо це не покращує продуктивність, не лематизуйте. Відмова від лематизації є консервативним підходом, і йому слід віддавати перевагу, якщо немає значного приросту продуктивності. Наприклад, популярний метод аналізу настроїв, VADER, має різні оцінки залежно від форми слова, тому введені дані не слід використовувати як корінь або лематизувати [8].

Застосування лематизації:

- використання в біомедицині: обробка тексту, пов’язаного з біомедициною, може бути ефективною за допомогою спеціальної лематизації та може підвищити ефективність завдань пошуку даних;
- використовується в комплексних системах пошуку;
- використовується в компактному індексуванні.

Реалізація лематизації слів за допомогою NLTK.

NLTK надає клас WordNetLemmatizer, який є тонкою обкладинкою, оберненою навколо wordnetCorpus. Цей клас використовує функцію під назвою

Morphy() для класу WordNetCorpusReader для пошуку кореневого слова/леми [13].

Сучасні підходи, керовані даними, зазвичай розглядають лематизацію як завдання класифікації, де класи представлені бінарними деревами редагування, створеними навчальними даними. Якщо дана пара лексем, їх бінарне дерево редагування створюється шляхом обчислення префікса та суфікса навколо найдовшої спільної підпослідовності та рекурсивної побудови дерева, доки не залишиться жодного спільного символу. Таким деревам редагування вдається охопити велику частку морфологічної закономірності, особливо для мов, які покладаються на суфіксацію для морфологічної флексії (наприклад, західноєвропейські мови), для яких такі методи були в першу чергу розроблені. На основі індукції дерева редагування були запропоновані різні лематизатори [14].

Перевагою лематизаторів над стемерами є той факт, що вони асоціюють різні форми слів з відповідними лемами з урахуванням частин мови, що дає більшу якість обробки слів.

1.4.2.5 POS-тегування

Додавання тегів до частини мови або POS-тегування – це процес позначення слова в тексті певною частиною мови на основі контексту та визначення. Простою мовою можна сказати, що тегування POS - це процес ідентифікації слів як іменників, займенників, дієслів, прикметників тощо.

Деякі слова можуть функціонувати кількома способами, якщо їх використовувати в різних обставинах. POS-тегування тут відіграє вирішальну роль, щоб зрозуміти, у якому контексті використовується слово в реченні. POS-тегування корисно для аналізу речень, пошуку інформації, аналізу настроїв тощо [15].

Існує чотири основні методи додавання тегів до POS:

1. Додавання тегів вручну: це означає, що люди, які знають правила синтаксису, застосовують тег до кожного слова у фразі. Це трудомісткий, старий неавтоматизований метод.

2. Тегування на основі правил: перший автоматизований спосіб додавання тегів. Складається з ряду правил (якщо попереднє слово є артиклем, а наступне слово є іменником, то воно є прикметником…). Це має робити фахівець.

3. Стохастичні/ймовірнісні методи: автоматичні способи призначення POS слову на основі ймовірності того, що слово належить до певного тегу, або на основі ймовірності того, що слово має відповідний тег на основі послідовності попередніх/наступних слів. Це найпопулярніші, найбільш використовувані та найуспішніші методи досі. Їх також простіше реалізувати (враховуючи, що уже є попередньо анотовані зразки — корпус). Серед цих методів можна визначити два типи автоматизованих імовірнісних методів: дискримінаційні ймовірнісні класифікатори (прикладами є логістична регресія, SVM та умовні випадкові поля — CRF) і генеративні ймовірнісні класифікатори (прикладами є наївні моделі Байєса та приховані моделі Маркова — HMM).

4. Методи глибокого навчання: методи, які використовують методи глибокого навчання для визначення тегів POS. Наразі ці методи не виявилися кращими від стохастичних/імовірнісних методів у додаванні тегів до POS — вони, щонайбільше, мають той самий рівень точності — ціною більшої складності/часу навчання [16].

1.4.3 Підходи для аналізу

1.4.3.1 Згорткові нейронні мережі

Згорткові нейронні мережі (CNN - convolutional neural network) можуть використовувати розподілені представлення слів, перетворюючи лексеми, що містять кожне речення, у вектор, утворюючи матрицю, яка буде використовуватися як вхідні дані [20]. Їх генеративні моделі все ще залишаються закритими, тобто вони можуть передбачати лише слова, які досить часто бачили в навчальних даних, тому конструкція CNN допомагає лише з рідкісними випадками розпізнавання нових слів. Крім того, побудова вкладень із написання для кожного токена неявно навчає функцію вбудовування на базі CNN «правильно вживати часті слова» замість того, щоб передбачати нові слова [21].

Проте результати, досягнуті за допомогою цієї порівняно простої архітектури CNN, є достатньо непоганими, і свідчать про те, що вона може служити заміною для добре встановлених базових моделей, таких як SVM або логістичної регресії [20].

На жаль, недоліком моделей на основі CNN – навіть простих – є те, що вони вимагають від практиків визначити точну архітектуру моделі, яка буде використовуватися, і встановити супутні гіперпараметри. На практиці налаштувати всі ці гіперпараметри просто неможливо, особливо тому, що оцінка параметрів потребує інтенсивних обчислень [20].

Але можна сказати, що ця мережа здатна з певними модифікаціями і використанням параметрів певної задачі співставляти речення для знаходження їх подібності.

1.4.3.2 Дерева розбору

Дерева аналізу послідовності широко використовуються в комп'ютерній лінгвістиці, де вони представляють складові аналізу речення природної мови, і в компіляторах, де вони забезпечують синтаксичну структуру вхідної програми. Вони виробляються за допомогою алгоритму аналізу на основі граматики [22]. Граматикою можна назвати набір терміналів та нетерміналів, а також правил продукції і аксіом.

Було запропоновано ряд підходів до вивчення дерев синтаксичного аналізу, включаючи ядро згортки на основі навчання опорних векторів (SVM), а також структурну подібність на основі прямого зіставлення дерев синтаксичного аналізу для речень [23]. При аналізі набору речень можна ввести таке поняття, як Parse Thicket (PT) — граф, який містить дерева синтаксичного аналізу для кожного речення, а також додаткові дуги для зв'язку між реченнями [23].

В роботі Boris Galitsky [23] можна побачити способи застосування дерев розбору для визначення зв'язності і подібності речень між собою. Для цих завдань основну роль грають висловлювання комунікативних дій, які дозволяють зв'язувати речення в певний послідовний ланцюг, розбиття речень на дерева за допомогою пошуку дієслівних конструкцій та використання POS, а також знаходження підграфів на основі послідовності слів або речень для встановлення подібності між реченнями чи абзацами.

Приклад дерев розбору речень наведено на рисунку 1.1.

Прикладом граматики може виступати комбінаторна категоріальна граматика (CCG) - лексикалізована теорія граматики, яка була успішно застосована до ряду проблем у NLP, включаючи створення дерева. Вона має бінарну природу розгалуження.

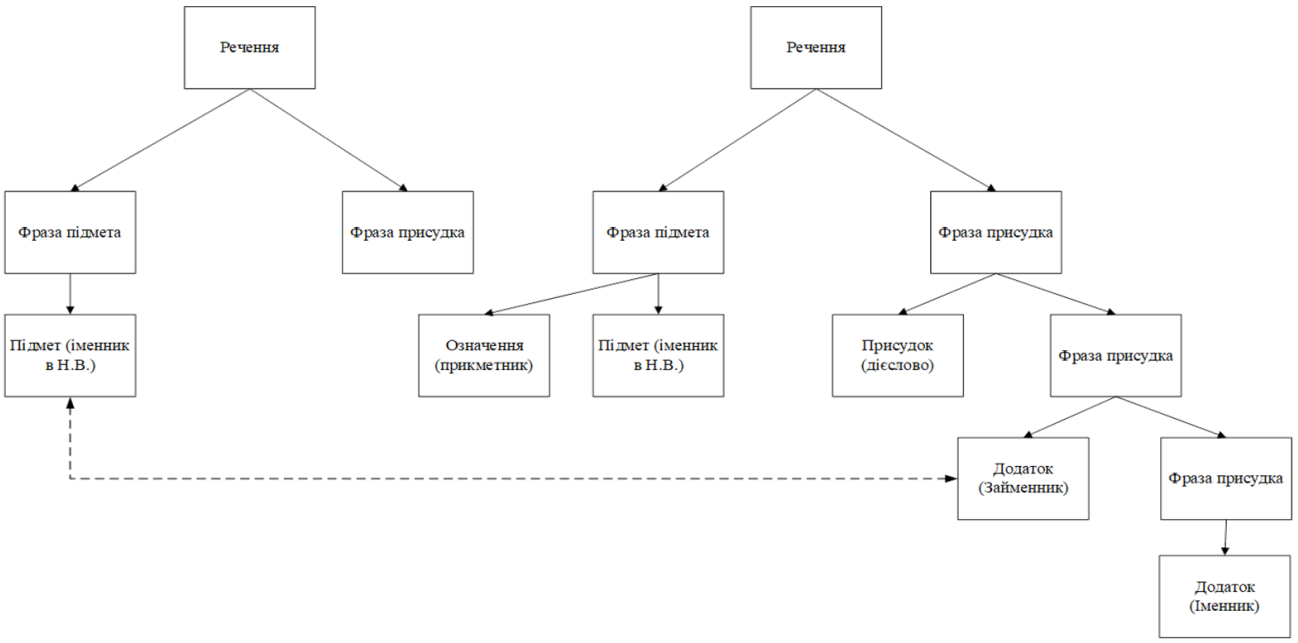

Рисунок 1.1 – Приклад дерев розбору речень

Під час синтаксичного аналізу CCG суміжні категорії можуть об’єднуватися за допомогою комбінаторних правил CCG. Наприклад, дієслівна фраза може об’єднатися з іменником, щоб виключити цей іменник з фрази. Категорії також можна об’єднувати за допомогою композиції функцій. На додаток до двійкових правил, таких як застосування функції та композиція, існують також унарні правила, які діють на одну категорію, щоб змінити її тип, наприклад підвищити рівень дерева, який розглядається.

Ресурс, який використовується для створення широкого охоплення - синтаксичний аналізатор CCG [24].

В загальному випадку можна виділити 2 підходи до формування дерев: зверху вниз та знизу вгору.

Рекурсивний аналізатор спуску - це різновид аналізатора зверху вниз. Синтаксичний аналізатор зверху вниз будує дерево аналізу зверху вниз, починаючи з початкового нетерміналу. Прогнозний аналізатор — це особливий випадок аналізатора рекурсивного спуску, де зворотне відстеження не потрібне. Ретельне написання граматики означає усунення лівої рекурсії та лівого

факторування, отримана граматика буде граматикою, яку можна проаналізувати рекурсивним синтаксичним аналізатором [25].

Однією з багатьох гілок низхідного аналізу є синтаксичний аналіз LL, або крайній лівий символ перегляду. Цей синтаксичний аналіз створить таблицю, у якій можна буде знайти правильну дію на основі символів у верхній частині стека та на початку введення. Основна ідея таблиці аналізу LL полягає в тому, щоб знайти набір термінальних символів, які можуть з'явитися на початку рядка, розширеного з правил продукції [26].

Синтаксичний аналіз зі зсувом і зменшенням (Shift-Reduce) є типом синтаксичного аналізу знизу вгору, оскільки він генерує дерево аналізу від листків (знизу) до кореня (вгору). Тут вхідний рядок зводиться до початкового символу. Це зменшення може бути досягнуто шляхом безпосередньої обробки крайнього правого похідного від початкового символу до вхідного рядка.

Синтаксичний аналізатор shift-reduce може виконувати чотири основні операції:

1. Зсув - ця операція передбачає переміщення поточного символу або слова з вхідного буфера в стек.
2. Зменшення - коли синтаксичний аналізатор знає, що права рука дескриптора знаходиться у верхній частині стеку, операція зменшення застосовує продукційні правила, тобто витягує RHS (правіші) продукційні правила зі стеку та виштовхує LHS (лівіші) продукційні правила на стек.
3. Прийняття - після повторення операцій зсуву та зменшення, якщо стек містить початковий символ вхідного рядка, а вхідний буфер порожній, вхідний рядок вважається прийнятим.
4. Помилка - якщо синтаксичний аналізатор не може виконати операцію зсуву або зменшення, а також рядок не приймається, то він перебуває в стані помилки [27].

1.4.4 Метрики

1.4.4.1 Косинусна подібність

Документ може бути представлений як вектор термінів, розміри вектору якого посилаються на терміни, доступні в документі. Значення розміру - це наявність терміну в документі. Документ можна описати як векторну форму наступним чином:

$$\vec{d} = (w_{d0}, w_{d1}, \dots, w_{dk})$$

Як і документ, послідовність термінів/інший документ можна описати як векторну форму так:

$$\vec{q} = (w_{q0}, w_{q1}, \dots, w_{qk}),$$

де w_{di} та w_{qi} ($0 <= i <= k$) є числами з плаваючою точкою, що вказують на частоту кожного терміну в документі, тоді як розмірність кожного вектору відповідає терміну, доступному в документі.

На основі векторної подібності подібність між двома векторами можна визначити як:

$$Sim(\vec{q}, \vec{d}) = \frac{\vec{q} * \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{k=1}^{t} w_{qk} * w_{dk}}{\sqrt{\sum_{k=1}^{t} (w_{qk})^2} * \sqrt{\sum_{k=1}^{t} (w_{dk})^2}}$$ [28].

Косинусна подібність N-грам вимірюється для визначення подібності послідовностей слів у реченнях. Порядок слів опосередковано визначає смислове значення речень [29].

Сама по собі косинусна подібність не завжди є ефективним засобом для знаходження подібності між реченнями, проте вона є гарним базисом для надбудов та покращень, що нададуть гарні результати. Приклади таких покращень можна знайти в роботі N. Adilah Hanin Zahri та ін. [29], де окрім косинусної подібності використовується коефіцієнт перекриття слів, з відображенням наявності тих самих слів в порівнюваних реченнях, в роботі Delphine Charlet та ін. [30], де включається в формулу матриця відношення між словами, в якій кожний елемент – значення відношення одного слова до іншого.

1.4.4.2 Відстань Левенштейна

Відстань Левенштейна - алгоритм, що використовується для ідентифікації матриці подібності між словами. Щоб визначити подібність речень, вимірювання подібності слів має більше значення. Відстань Левенштейна підраховує мінімальну кількість подібностей, необхідних для операції вставки, видалення та модифікації кожного символу, який може вимагати перетворення з речення в інше речення. Ступінь спорідненості допомагає створювати кращий підсумок тексту шляхом аналізу подібності тексту. Ступінь вимірювання може бути слово-слово, слово-речення, речення-слово і речення-речення [31].

Алгоритм для програмного обрахунку відстані Левенштейна:

Крок 1: Ініціалізація

- Встановити n як довжину першого слова, встановити m як довжину другого слова.
- Побудувати матрицю, яка містить m рядків і n стовпців.
- Проініціалізувати рядки нульовими значеннями.
- Проініціалізувати стовпці на 0..m.

Крок 2: Обробка

- Пройти за стовпцями (s[i]) через матрицю.
- Для кожного стовпця пройти за рядками(t[j]).
- Якщо s[i] дорівнює t[j], вартість дорівнює 0.
- Якщо s[i] не дорівнює t[j], вартість дорівнює 1.
- Встановіть комірку d[i,j] матриці рівною мінімуму:

а) комірка безпосередньо над поточною + 1: d[i-1,j] + 1;

б). комірка ліворуч від поточної + 1: d[i,j-1] + 1;

в) комірка по діагоналі вище та ліворуч + вартість: d[i-1,j-1] + вартість.

Крок 3: Результат

Крок 2 повторюється, доки не буде знайдено значення d[n,m] [32].

1.4.4.3 TF-IDF

TF-IDF — це числова статистика, яка показує релевантність ключових слів певним документам або, можна сказати, надає ті ключові слова, за допомогою яких деякі конкретні документи можна ідентифікувати або класифікувати. TF-IDF – це комбінація двох різних слів, тобто частота терміну та інверсна частота документа.

По-перше, термін «Частота терміну» (TF) використовується для вимірювання того, скільки разів термін присутній у документі. Дуже добре відомий факт, що загальна довжина документів може варіюватися від дуже маленької до великої, тому існує ймовірність того, що будь-який термін може зустрічатися частіше у великих документах порівняно з маленькими. Отже, щоб вирішити цю проблему, кількість однакових термінів w_i у документі ділиться на загальну кількість термінів у цьому документі, щоб знайти частоту термінів.

$$TF_i = \frac{Count(w_i)}{\sum_{j=1}^{n} Count(w_j)}$$

де w_i – i-й термін, w_j – j-й термін

Інверсна частота документів (IDF). Під час обчислення частоти термінів у документі можна помітити, що алгоритм обробляє всі ключові слова однаково, не має значення, якщо це стоп-слово, наприклад «of», що є неправильним. Усі ключові слова мають різну важливість. Зворотна частота документа призначає меншу вагу частим словам і призначає більшу вагу словам, які є рідкісними.

$$IDF_i = ln(\frac{Count(doc)}{Count(doc\ with\ w_i)})$$

де *Count(doc)* – кількість документів, що розглядається, *Count(doc with* w_i*)* – кількість документів, що містять термін w_i з набору всіх документів.

TF-IDF — це не що інше, як множення термінової частоти (TF) і зворотної частоти документа (IDF) [33].

1.5 Існуючі програмні рішення

В цілому можна сказати, що відомих програмних рішень, які б повноцінно розв’язували представлену проблему немає, або ж вони є спеціалізованими і створені для локального застосування. Найближчим аналогом для розроблюваної системи можна навести інструменти для перевірки на плагіат, оскільки вони виконують зворотню задачу. Далі буде коротка характеристика деяких з них.

1.5.1 Plagiarism Checker X

Plagiarism Checker X 2023 допоможе перевірити наявність плагіату в наукових роботах, блогах, завданнях та веб-сайтах. Завдяки більш високій швидкості і точності можна легко перевірити схожість тексту всього за кілька секунд.

Plagiarism Checker X допомагає студентам, перевіряючи проблеми з дублюванням у їхніх завданнях та статтях. Ви можете легко перевірити схожість вмісту та знайти першоджерела, правильні цитати та, як наслідок, отримати кращі оцінки.

Це програмне забезпечення призначене для перегляду найпопулярніших форматів файлів, включаючи документи Microsoft Word, електронні таблиці, PDF, RTF та звичайний текст, і сумісне з Windows 11, 10, 8, 7 та Vista.

Програма перевірки подібності вмісту доступна англійською, іспанською, французькою, німецькою, італійською, голландською та португальською мовами. Незабаром з'являться нові мови.

Порівняння тексту двох документів, щоб визначити схожість. Ця функція виділяє як оригінальний, так і альтернативний контент скрізь, де знаходить дублікати.

Автори вмісту веб-сайтів, блогери та видавці можуть використовувати цю безкоштовну програму перевірки плагіату для аналізу якості онлайн-вмісту, щоб уникнути штрафних санкцій з боку пошукових систем.

Додаток надає функції виключення цитат і бібліографії для допомоги академічним користувачам, щоб вони могли налаштувати сканований контент для звітів відповідно до вимог свого інституту.

Всебічна звітність у форматі PDF / DOCX є відмінною рисою програмного забезпечення для боротьби з плагіатом. Звіти про оригінальність виділені кольором, що вказує на рівень підібраної подібності [34].

1.5.2 Copyscape

Мільйони власників веб-сайтів довіряють Copyscape перевіряти оригінальність свого нового вмісту, запобігати дублюванню вмісту та шукати копії існуючого вмісту в Інтернеті.

Copyscape надає безкоштовну перевірку плагіату для пошуку копій веб - сторінок в Інтернеті, а також два потужні професійні рішення для запобігання шахрайству з вмістом.

Copyscape Premium забезпечує більш потужне виявлення плагіату, ніж безкоштовний сервіс, плюс безліч інших функцій, включаючи перевірку оригінальності при копіюванні-вставці, завантаження файлів PDF і Word, пакетний пошук, особистий Індекс, відстеження звернень, API і інтеграцію з WordPress.

Copysentry забезпечує комплексний захист веб-сайту, автоматично скануючи веб-сторінки щодня або щотижня та надсилаючи електронне повідомлення, коли виявляються нові копії вмісту веб-сайту.

Copyscape також пропонує безкоштовні банери попередження про плагіат для веб-сайту, щоб застерегти потенційних плагіаторів від крадіжки його вмісту, безкоштовний інструмент для порівняння двох веб-сторінок або статей та вичерпний посібник із боротьби з плагіатом [35].

1.5.3 Turnitin

Turnitin-це програмне забезпечення, орієнтоване на цілісність, що є однією з причин його вражаючої популярності в академічному середовищі.

Інструмент дозволяє швидко виявити плагіат, порівнюючи набори контенту з найбільшою колекцією оцифрованих академічних матеріалів і виділяючи подібності.

Інструмент подібності, наданий Turnitin checks, пропонує вичерпні звіти про плагіат. Він порівнює наданий текст з активними та архівованими веб-сторінками, роботами, включеними до сховища Turnitin, періодичними виданнями, журналами та іншими публікаціями.

Звіти, згенеровані цим інструментом, містять детальну інформацію про результати перевірки на плагіат. Оцінка подібності - це відсоток плагіату у перевіреній статті.

Turnitin Similarity класифікує плагіат-контент за двома категоріями: збіги і джерела. Збіги - це тексти, які не повністю ідентичні, але дуже схожі. Джерелами є ідентичні фрагменти тексту, взяті з інших статей.

Вони можуть бути дуже корисними, тому що потрібні різні рішення для виправлення роботи. Джерело можна виправити за допомогою правильної цитати, тоді як збіг вимагає подальшого читання та більш оригінального вмісту.

Turnitin Similarity - чудовий вибір для студентів, оскільки він сумісний з кількома інструментами навчання, такими як Blackboard learn, Brightspace, MS Teams та Moodle.

Turnitin - це ліцензійне програмне забезпечення, призначене для академічних установ. Його не можливо отримати як індивідуальний користувач, якщо школа ним не користується.

Якщо ви представляєте установу і хочете ним скористатися, це можливо зробити, заповнивши форму запиту.

Давайте швидко розглянемо його ключові особливості:

- ефективне рішення для виявлення плагіату;
- впровадження авторства на основі аналізу, заснованого на даних;
- функції зворотного зв'язку і виставлення оцінок для полегшення навчального діалогу;
- освітні, творчі ресурси для підвищення академічних навичок (блоги, офіційні документи та багато іншого) [36].

1.5.4 Unicheck

Завдяки поєднанню надсучасних технологій та інтуїтивного дизайну, Unicheck допомагає підвищити якість оригінальних текстів, а не просто вказує на текстові збіги.

Розробники створюють антиплагіатний сервіс згідно побажань та пропозицій користувачів, тому цей програмний продукт отримує схвальні відгуки по всьому світу. Сервіс Unicheck вже успішно використовують в США, Латинській Америці, Іспанії, Бельгії, Україні, Австралії та інших країнах.

Перевірка на плагіат в особистих цілях:

- підтримка 99% файлових форматів та масового завантаження файлів;
- детальний звіт для виявлення та знешкодження плагіату;
- можливість скористатися розширенням для Google і перевіряти роботи на плагіат будь-де.

Перевірка на плагіат для навчання:

- легка інтеграція з усіма основними LMS системами;
- надшвидкий пошук в Інтернеті та персональній бібліотеці користувача;
- посилання на текстові збіги без вірусів, фішингу та інших загроз для безпеки.

Перевірка на плагіат у пакеті Індивідуальний+:

- можливість здійснювати перевірку на плагіат у своїй діловій екосистемі;
- структура тексту та форматування залишаються незмінними;
- агрегована аналітика для поглибленого аналізу.

Unicheck знаходить текстові збіги та розпізнає маніпуляції з текстом за лічені хвилини. Програма автоматично генерує онлайн звіт, який можна використовувати для вдосконалення письмових навичок студентів.

Unicheck - це комплексний онлайн сервіс для запобігання плагіату. Усі нові функції програми, що вдосконалюють пошук, вбудовуються у звіт. Жодних додаткових інструментів!

Modifind - технологія, що виявляє підозрілі маніпуляції з текстом. Якщо такі знайдуться, інформація про них з'явиться в окремій вкладці звіту Unicheck.

Unicheck обробляє персональні дані відповідно до політик конфіденційності FERPA, GDPR та COPPA. Більше того, для кожного навчального закладу створюється окреме сховище даних.

Завдяки хостингу AWS, Unicheck збільшив час роботи до 99,95%. Отже, тепер є можливість користуватися сервісом без раптових збоїв в системі навіть під час високих навантажень.

Перевіряючи академічні роботи з Unicheck, ви отримуєте звіт онлайн впродовж декількох хвилин. За продуктивністю програми пильно стежить команда розробників [37].

1.5.5 Quetext

Плюси:

- є чіткий і детальний звіт;
- вбудований помічник цитування допомагає додавати відсутні цитати;
- документи не зберігаються в базі даних;
- доступні варіанти підтримки (але немає підтримки в реальному часі).
- Мінуси:
- часткові збіги і помилкові спрацьовування;
- щомісячна підписка на суму не менше 9,99 доларів США після безкоштовної пробної версії;
- не ефективний для наукових джерел;

- виявляє не весь плагіат.

Теоретично безкоштовний, Quetext пропонує безкоштовну пробну версію з п'яти "сторінок" - при цьому сторінка вважається рівною 500 словами. Однак було виявлено, що після сканування одного документа з 800 слів сайт вже повідомив, що безкоштовна пробна версія закінчилася, і більше не буде приймати жодних документів, якими б короткими вони не були.

Крім того, завантаження файлів у Quetext - це професійна функція. Якщо не заплатити, то доведеться копіювати та вставляти текст із документа.

Платна версія коштує щонайменше 9,99 доларів на місяць (доступні дорожчі плани), що дозволяє перевірити до 100 000 слів.

Якість збігів. Quetext виявляє більше плагіату (в середньому 57%), ніж будь-який безкоштовний інструмент, але він не в змозі повністю зіставити кожен вихідний текст з одним джерелом. Різні пропозиції приписуються різним джерелам, що призводить до багатьох хибнопозитивних результатів.

Quetext стверджує, що перевіряє веб-сторінки та академічні джерела, але на практиці було виявлено, що він погано справляється з виявленням плагіату з академічних джерел.

Зручність використання. Процес сканування в Quetext відбувається досить повільно. Як тільки це закінчиться, звіт про плагіат стане простим для розуміння та надасть корисний огляд. Показаний точний відсоток з виділенням подібностей в тексті.

Різні джерела не виділяються різними кольорами. Натомість помаранчеве підсвічування використовується для часткових збігів, Червоне - для повних збігів. Натискання на виділене речення показує відповідний оригінальний текст, а інший відсоток вказує на те, наскільки пропозиція була схожа на оригінал.

Надійність. Quetext стверджує, що він не зберігає текст, що перевіряється, у базі даних і що весь надісланий текст залишається приватним та зашифрованим. [38].

Засоби перевірки на плагіат в своїй більшості надають засоби для порівняння текстів, проте кожен з них має певні обмеження. До того ж оскільки вони намагаються знайти елементи плагіату, а в даній роботі ми намагаємось вирішити зворотню задачу, що може вплинути на акценти та може спричинити використання різних підходів. До того ж вони намагаються бути багатомовними, а це може спричинити погіршення якості для маловживаних та не рідних мов.

1.6 Задача випускної кваліфікаційної роботи

Метою роботи є створення за допомогою підходів NLP та нейромережевих технологій інтелектуальної системи для підтримки прийняття рішень при оцінюванні відповідей студентів на відкриті питання контрольних заходів та аналіз її ефективності.

Об’єктом роботи є процес порівняння текстів на їх лексичну, синтаксичну та семантичну подібність за допомогою інструментів NLP та нейромережевих технологій.

Предметом роботи є система для аналізу та порівняння текстів за допомогою засобів NLP та нейромережевих технологій.

1.7 Вимоги

Системні вимоги (рекомендовані):

- процесор: IntelCore i7-9750H;
- оперативна пам'ять: 8Gb;
- місце на диску: 1 Gb;
- відеокарта Intel HD Graphics 620;
- ОС: Windows 10.

Функціональні вимоги:

1. Перевірка обраного тексту на відповідність еталонному варіанту.
2. Порівняння обраного тексту з введеною базою текстів на схожість.
3. Порівняння обраного тексту з корпусом текстів на схожість.
4. Визначення рівня схожості.
5. Виділення областей схожості за рівнями.
6. Виведення оцінки.
7. Наявність інтерфейсу.

Нефункціональні вимоги:

1. Оптимізація роботи викладача зі збереженням якості оцінювання.
2. Робота без перебоїв та помилок.
3. Видача результату за припустимий час.
4. Зручний та простий у використанні інтерфейс.

1.8 Система як «чорна скриня»

Вхідними даними буде відповідь студента в текстовому варіанті та еталонний варіант на початку роботи.

Вихідними даними буде текст з позначеннями результатів порівнянь та оцінка роботи.

«Чорна скриня» системи зображена на рисунку 1.2.

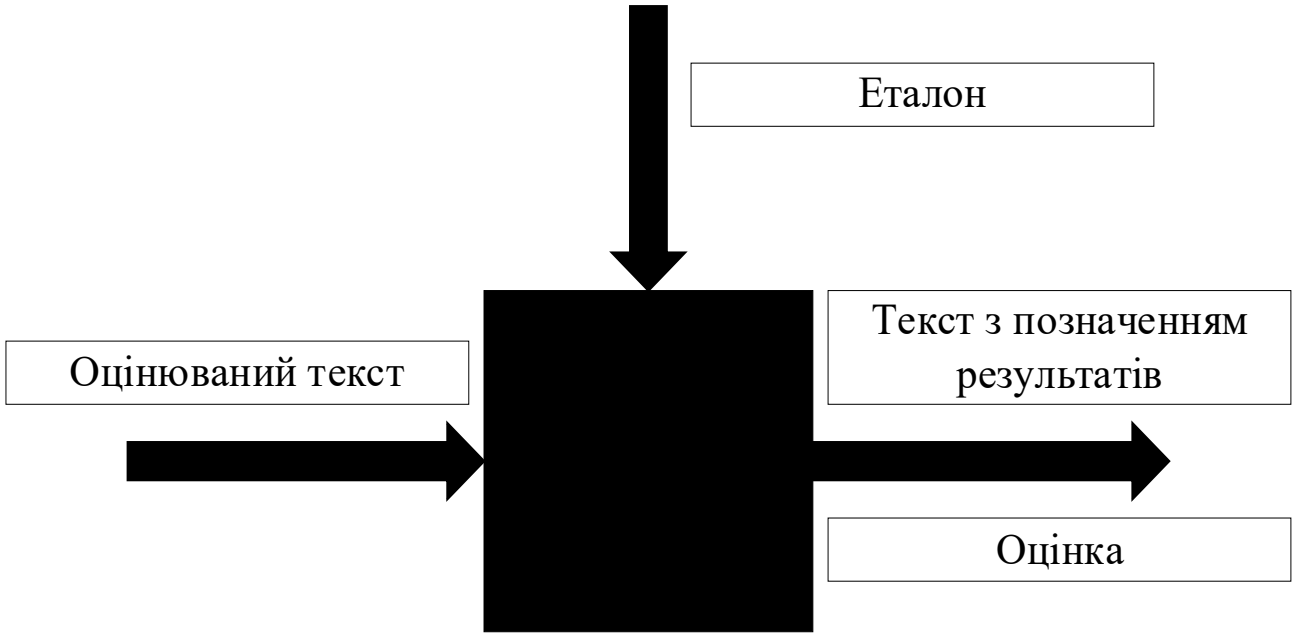

Рисунок 1.2 – Система як «чорна скриня»

1.9 Висновки до першого розділу

Галузь освіти переходить в цифровий простір і вона потребує певних вдосконалень, таких як автоматична оцінка робіт студентів за допомогою можливостей обчислювальних машин. На даний момент існуючі програмні рішення є не досконалими і їм бракує деякого функціоналу для задоволення потреб викладачів, тому вони не використовуються. І хоча існує багато підходів до рішення подібного роду задач за допомогою інструментів NLP, проте створення та впровадження ефективної системи потребують неабияких зусиль. А існуючі засоби мало орієнтовані на цю проблему через її вузьку спеціалізацію.

Тож я вважаю, що ця проблема є доволі актуальною, і метою роботи буде використання деяких з наведених інструментів та підходів NLP для побудови відповідної системи і дослідження її ефективності.

2 Проектні рішення для інтелектуальної системи для прийняття рішень при оцінюванні відповідей студентів на відкриті питання контрольних заходів

2.1 Архітектура інформаційної технології

Система складається з наступних модулів:

1. Модуль інтерфейсу користувача, який приймає вхідні дані, викликає виконання алгоритмів та отримує результат, за допомогою запитів та команд.
2. Модуль для порівняння оцінюваного тексту із завантаженим еталонним варіантом на ідентичність комбінацій послідовних слів.
3. Модуль для порівняння оцінюваного тексту з базою завантажених текстів на ідентичність комбінацій послідовних слів.
4. Модуль для порівняння оцінюваного тексту із завантаженим еталонним варіантом на синонімічність слів та їх комбінацій.
5. Модуль для порівняння оцінюваного тексту з базою завантажених текстів на синонімічність слів та їх комбінацій.
6. Модуль для тренування нейронної мережі
7. Модуль для прогнозування синонімічності слів та їх конструкцій за допомогою нейронної мережі.
8. Модуль для визначення оцінки на основі результатів роботи інших модулів.

Модель зв’язків системи відображено на рисунку 2.1.

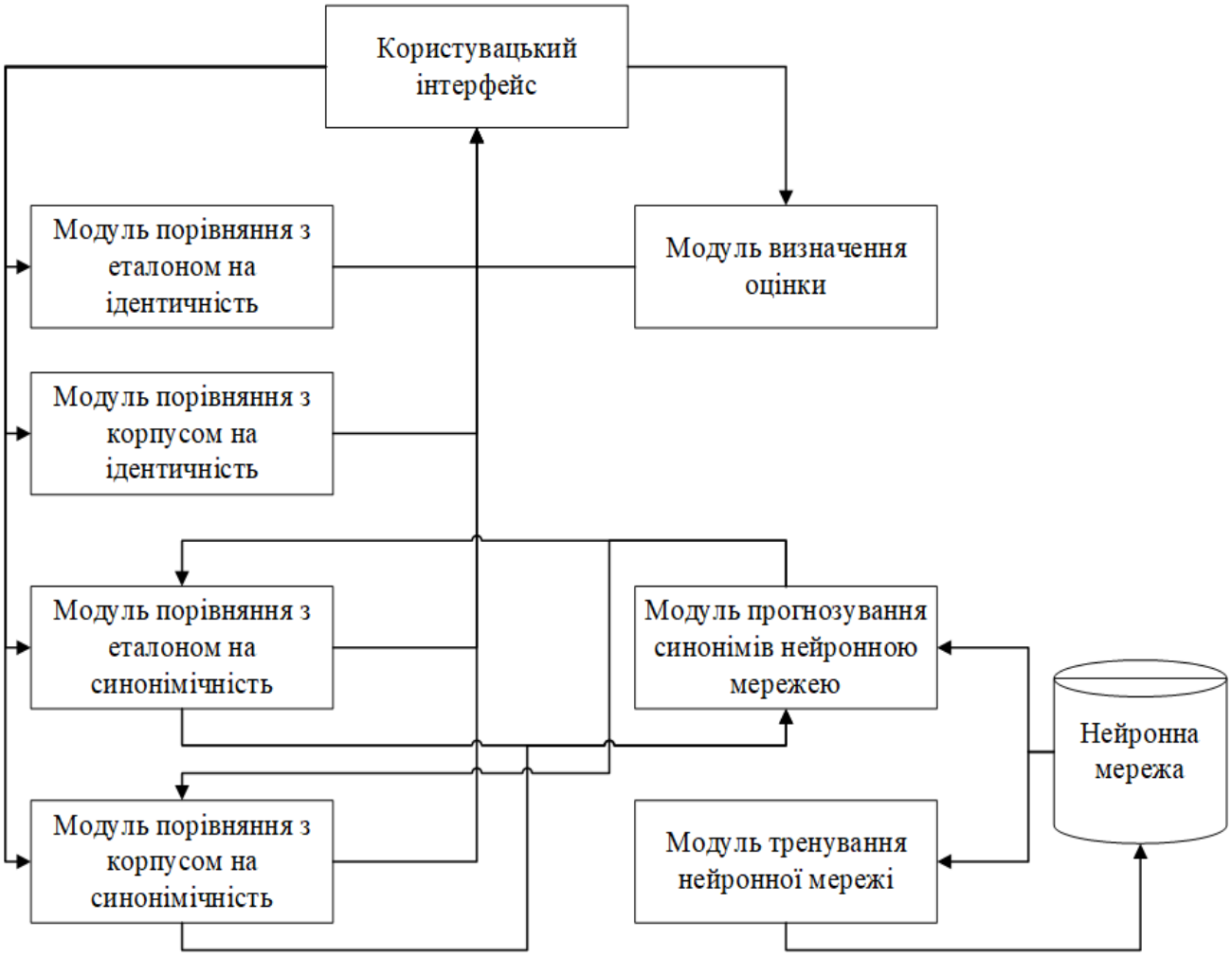

Рисунок 2.1 – Модель зв’язків системи

Технічна архітектура системи включає в себе наступні елементи:

- робоча станція, системні вимоги якої наведені вище;
- маршрутизатор;
- мережеві кабелі;
- монітор;
- маніпулятор «миша»;
- клавіатура.

Аналіз функцій системи наведено на рисунку 2.2

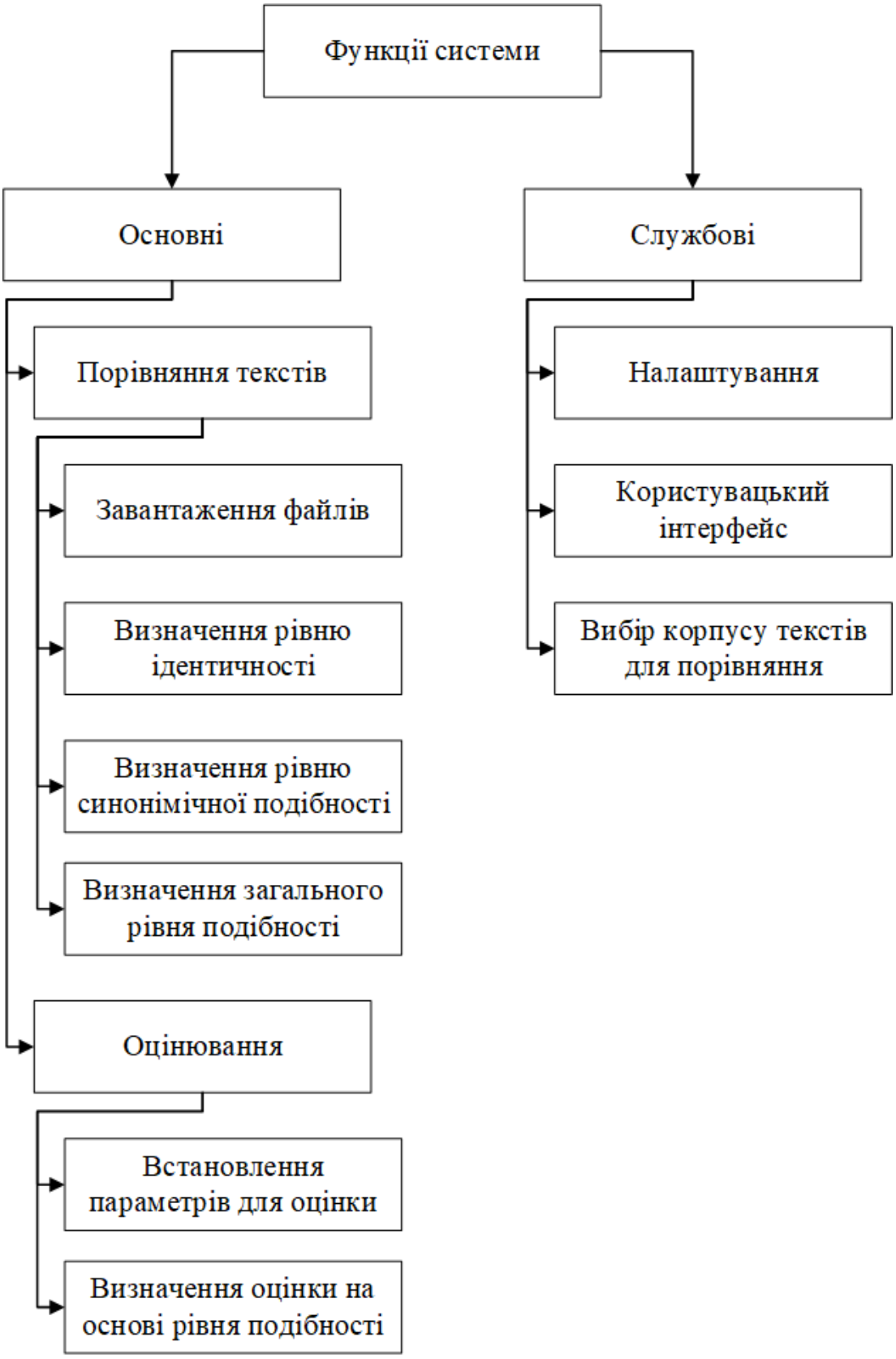

Рисунок 2.2 – Дерево функцій системи

2.2 Формалізація вихідних даних та встановлення логіко-математичних зв’язків, вибір критеріїв, обмежень

Результатом роботи програмного застосунку повинно бути відображення в користувацькому інтерфейсі двох наборів текстових даних, що відповідають двом варіантам відповіді на запитання – еталонному та тому, що проходив перевірку, з відзначенням певним кольором областей з різнем рівнем подібності: зелений для областей ідентичності еталонного варіанту та тексту, що перевіряється, жовтий для областей ідентичності текстів з корпусу та тексту, що перевіряється, блакитний для областей синонімічності між еталонним варіантом та текстом, що перевіряється, помаранчевий для областей синонімічності між текстами з корпусу та текстом, що перевіряється. Також в результуючому вікні повинна бути відображена очікувана/рекомендована оцінка.

Критеріями для виконання роботи є:

- еталонний варіант відповіді;
- варіант відповіді, що перевіряється;
- встановлені в налаштуваннях параметри для роботи алгоритму.

Обмеженнями є:

- обидва варіанти відповіді написані українською мовою;
- обмеження в розмірах на основі еталону.

2.3 Логіка програми

Основна мета полягає в тому, щоб співставити відповідь студента з еталонним варіантом відповіді викладача з використанням методів для порівняння тексту на схожість для визначення потенційної оцінки за виконану студентом роботу.

Початкове вікно має функції старту роботи програми, налаштувань та завершення роботи програми.

У вікні налаштувань можна підібрати необхідні параметри для алгоритму.

Після початку роботи програми, відображається вікно для завантаження файлів: еталону та тексту, що перевіряється. Після того, як тексти завантажені користувач підтверджує свій вибір, чим запускає роботу алгоритму.

Першим кроком в алгоритмі є попередня обробка даних (вилучення займенників, стоп-слів, деякої пунктуації та ін.), розбиття текстів на токени та їх лематизація.

Далі алгоритм використовує дворівневе порівняння. Перше – порівняння на ідентичність. Для цього застосовуються n-грами. Друге – порівняння на синонімічність. Для цього буде використана нейронна мережа.

Після порівняння з еталоном, базою завантажень та текстами в наявному корпусі формується рівень схожості.

В фіналі з урахуванням результатів порівняння виводиться оцінка та показуються рівні подібності в тексті.

2.4 Алгоритми, дослідження можливостей використання існуючих програмних засобів обчислювальної техніки

2.4.1 N-грами

N-грама - це послідовність із N слів: 2-грамма (яку ми називатимемо біграмою) — це послідовність двох слів, і 3-грамма (триграма) є послідовністю із трьох слів, і так далі. Через деяку термінологічну неоднозначність ми зазвичай відмовляємося від слова «модель» і використовуємо термін N-грама (і біграма тощо), що означає або саму послідовність слів, або прогнозну модель, яка призначає ймовірність для цієї послідовності. Хоча моделі n-грам набагато

простіші, ніж сучасні нейронні мовні моделі, засновані на RNN і трансформаторах, вони є важливим інструментом для розуміння фундаментальних концепцій моделювання мови [17].

Основна ідея алгоритму N-грам полягає у виконанні операції ковзного вікна розміром N у байтах над вмістом тексту для формування послідовності байтових фрагментів довжиною N. Кожен сегмент байта називається грамом. Підраховується частота появи разом з поточним всіх інших грамів і фільтрується відповідно до попередньо встановленого порогу, щоб сформувати список ключових грамів, який є векторним простором ознак цього тексту. Грам — це розмірність вектора ознак. Модель базується на припущенні, що поява N-го слова пов’язана лише з попередніми словами N-1, але не з будь-якими іншими словами. Ймовірність усього речення є добутком ймовірності появи кожного слова [18].

N-грами можна використати, оскільки передбачається, що підмножина оцінюваного тексту, яка охоплює більшу частину всіх N-грам, призведе до вищої якості системи [19].

2.4.2 Буквальний аналіз. Підхід на основі n-грам

Як було описано в розділі 1, n-грами часто використовуються для аналізу тексту як основний або допоміжний засіб. В нашому випадку вони будуть основою підходу для буквального (точного) порівняння текстів, що розглядаються.

Після того як була виконана попередня обробка текстів і були отримані токени (в нашому випадку слова, що не знаходяться в словнику стоп-слів), формується список n-грам.

Кожен елемент списку розширюється, шляхом додавання інформації про кожну відповідну n-граму (наприклад в якій позиції в тексті знаходиться відповідна n-грама, який тип подібності їй відповідає та інше). Сформувавши

відповідні списки для еталону та для тексту, що перевіряється, розпочинається порівняння цих списків.

Спочатку визначається чи ідентичні n-грама з еталонного варіанту та з того варіанту, що перевіряється. Для цього порівнюються слова в відповідних позиціях n-грам. У випадку, якщо хоча б одна з пар слів не співпадає, порівняння пари n-грам завершується і починається наступне порівняння. Слід зазначити, що для збільшення потенційних пар n-грам можна ввести словник замін слів (у випадку, якщо хоча б одне з них може бути замінене на інше в будь-який момент часу) або додати пропуск першої букви (вона може бути взаємозамінним префіксом).

Наступним етапом є вибір потенційно найкращої n-грами, що відповідає n-грамі в еталоні. Оскільки може бути ситуація, що одній n-грамі в еталоні відповідає декілька n-грам в тексті, що перевіряється (або навпаки), необхідно визначити, яка з n-грам є потенційно найкращим вибором. Для цієї задачі будемо застосовувати жадібний підхід, що полягає в визначенні чи оточена поточна n-грама іншими n-грамами, що знайшли свої пари в іншому тексті. Виконується прохід по списку в обидві сторони від n-грами, поки не буде досягнуто n-грам, що не знайшли своїх аналогів в іншому тексті. Таким чином формується список із потенційних кандидатів n-грам з іншого тексту для формування пари з поточною n-грамою. А оскільки ми сортуємо кандидатів від більшого до меншого – це і означає жадібний підхід.

Після того, як були визначені всі потенційні кандидати, виконується останній прохід за списком n-грам та формуються пари із потенційних кандидатів. Якщо n-грама має не одного кандидата, то обирається найкращий (з найбільшою кількість n-грам навколо), який після цього видаляється зі списку, даючи можливість наступному кандидату бути обраним наступною аналогічною n-грамою з іншого тексту.

Спочатку цей підхід застосовується для порівняння з еталонним варіантом відповіді і позначенням всіх знайдених n-грам відповідною відміткою в списку. Після цього алгоритм застосовується до корпусу (або корпусів) текстів, де перевіряється кожний з текстів та знайдені n-грами також позначаються відповідною відміткою. Слід зазначити, що процес визначення оточення для використання жадібного вибору найкращого кандидата використовується тільки для n-грам, притаманних цьому ж тексту з корпусу, з якого і поточна n-грама.

Таким чином у сформованому списку n-грам визначаються ті n-грами, що належать еталону або корпусу текстів без будь-яких змін в них, що і є результатом першої частини роботи алгоритму.

Узагальнена структура алгоритму буквального аналізу представлена на рисунку 2.3.

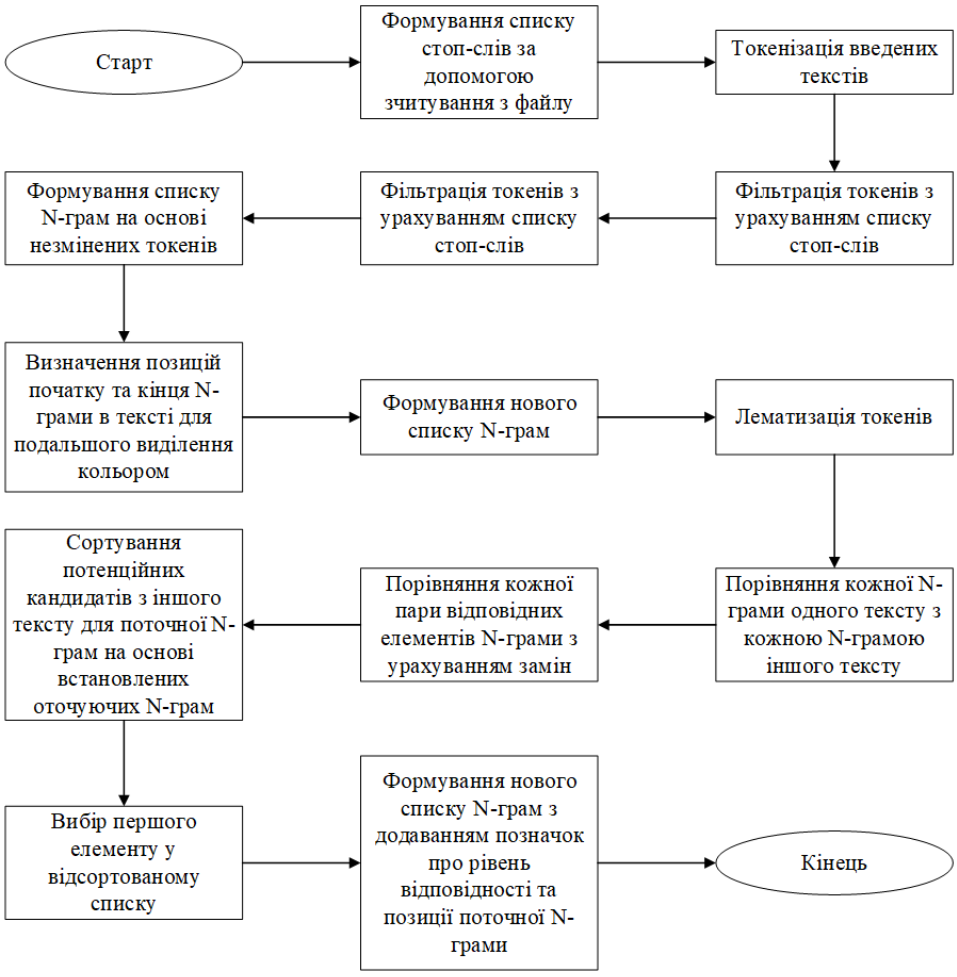

Рисунок 2.3 – Узагальнена структура алгоритму буквального аналізу

2.4.3 Аналіз на синонімію. Штучна нейронна мережа

Другий етап роботи алгоритму представляє собою визначення синонімії між n-грамами, що не були знайдені в еталоні та текстах корпусу при використанні буквального аналізу. Синонімія між n-грамами представляє собою синонімію між їх відповідними словами. Синонімія між словами в свою чергу представляє собою однаковість контексту, в якому ці слова використовуються. І для знаходження того, чи є слова синонімами на основі їх контексту буде використана нейронна мережа.

Штучні нейронні мережі – це математичні об’єкти, створені за моделлю існуючих біологічних нейронів, які знаходяться в мозку. Усі математичні моделі базуються на базовому блоці, відомому як штучний нейрон [39].

В даній роботі буде використана нейронна мережа, побудована на базі простого перцептрона з великою кількістю входів та виходів з додаванням активаційної функції в шар проміжних виходів, функції нормалізації softmax та похибки, розрахованої за допомогою крос-ентропії. Відповідна нейронна мережа представлена на рисунку 2.4.

Перший шар нейронної мережі представлений у вигляді n-ї кількості входів, де n – кількість слів в словнику нейронної мережі. Аналогічним є кількість проміжних виходів (або результатів активаційних функцій) та фінальних результатів, утворених за допомогою функції softmax.

Для того, щоб ефективно працювати з вхідними даними, треба їх перетворити на числове представлення. Тому на вхід нейронної мережі подається вектор, що складається із 0 та 1, де 1 – це слова, що оточують поточне слово в певному заданому радіусі.

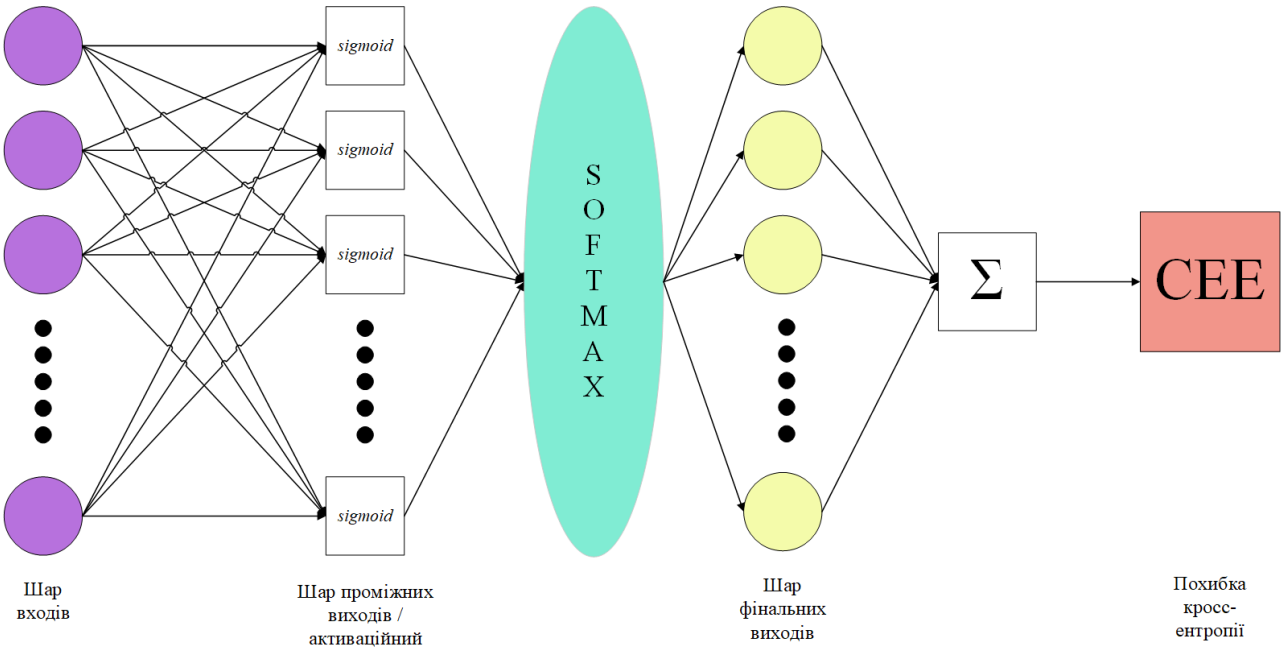

Рисунок 2.4 – Модель нейронної мережі

Після цього виконується передача значень входів в активаційні функції, кожна з яких відповідає одному слову в словнику. Значення входів множаться на відповідну вагу, яких у кожного входу n-на кількість. Тому матриця ваг нейронної мережі може бути представлено наступним чином:

$$weights = [w_{11} \cdots w_{1n} \vdots \ddots \vdots w_{1n} \cdots w_{nn}],$$

де n – розмір словника, w_{ij} – вага зв'язку між входом i та нейроном j з шару проміжних виходів.

Далі отримані значення на входах активаційної функції підсумовуються і розраховується результат застосування активаційної функції. Для цієї нейронної мережі була обрана логістична активаційна функція (або сигмоїда). Вона виглядає наступним чином:

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

ЇЇ перевагою є те, що вона є неперервно диференційованою, а також нормалізує значення до проміжку (0:1). Оскільки при навчанні ми хочемо дотримуватися напрямку градієнта, щоб знайти мінімум функції похибки, важливо, щоб не існувало областей, у яких функція помилки була б абсолютно плоскою. Оскільки сигмовид завжди має додатну похідну, нахил функції помилки забезпечує більший або менший напрямок спаду, який можна дотримуватися [40]. До того ж її похідну по відношенню до *x* можна представити у доволі простому вигляді:

$$\frac{d\sigma(x)}{dx} = \sigma(x) * (1 - \sigma(x)) \text{ [41]}, \quad (1)$$

що є дуже зручним при навчанні мережі при зворотньому проході.

Наступним етапом є ще один рівень нормалізації – застосування функції softmax. Вона виглядає наступним чином:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}},$$

де x_i – результат і-го виходу, а $\sum_{j=1}^{n} e^{x_j}$ – сума всіх виходів.

Застосувавши цю функцію ми отримуємо значення, що можна інтерпретувати як відсоткові ймовірності активації кожного з виходів.

Після того, як було отримано результат функції softmax необхідно знайти значення втрат – відхилення отриманого значення від очікуваного. Для цього можна використати функцію втрат крос-ентропії. Формула для цієї функції виглядає наступним чином:

$$loss = -\sum_{i=1}^{n} Target_i * log\ log\ (Predicted_i),$$

де $Target_i$ – те, що ми хочемо отримати на виході з нейронної мережі, а $Predicted_i$ – те що отримали.

Оскільки в даній роботі ми маємо велику кількість вхідних векторів, то в якості помилки маємо середнє значення всіх помилок утворених від входів (кожний вхідний вектор дає одне значення похибки).

Після того, як ми отримали похибку, нашою задачею є її мінімізація. Для цього ми використаємо метод проходу по мережі в зворотньому напрямку з метою коригування ваг. Це метод зворотнього поширення похибки (Back Propagation). Алгоритм зворотного поширення шукає мінімум функції помилки у ваговому просторі за допомогою методу градієнтного спуску. Комбінація вагових коефіцієнтів, яка мінімізує функцію помилки, вважається рішенням проблеми навчання. Оскільки цей метод вимагає обчислення градієнта функції помилки на кожному кроці ітерації, ми повинні гарантувати безперервність і диференційованість функції помилки [40]. Тут і проявляється перевага логістичної функції, тому вона є однією з найбільш популярних функцій активації для мереж зворотного поширення [40].

На скільки ми повинні змінити кожну вагу? Однією природною відповіддю є: пропорційно його впливу на помилку; чим більший вплив ваги *w*, тим більше зменшення помилки, яке може бути спричинене його зміною, і, отже, тим більшу зміну повинен зробити наш алгоритм навчання у цій вазі [42].

Для визначення значення, на яке треба змінити певну вагу, необхідно обрахувати значення похідної помилки по відношенню до поточного значення ваги. Оскільки в більшості випадків нейронна мережа складається не з одного шару, то для визначення рівню впливу тієї чи іншої ваги на значення похибки необхідно використовувати правило ланцюга (chain rule), яке звучить наступним чином: однією з можливих композицій функцій g і f є складена функція $f[g(x)]$, значення якої існує для всіх x в області g, так що всі значення $g(x)$ знаходиться в області f. Грубо кажучи, складена функція $f[g(x)]$ отримує «вихід» $g(x)$ і

використовує його як «вхід» функції *f(x)*. Ланцюгове правило використовується для знаходження похідної складеної функції *y = f[g(x)]*, де *y = f(u)* і *u = g(x)*, і визначається як

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} * \frac{\partial u}{\partial x} \quad [43]$$

У випадку даної роботи застосування правила ланцюга можна представити наступним чином:

$$\frac{\partial CEE}{\partial w_{i,j}} = \frac{\partial CEE}{\partial SM_j} * \frac{\partial SM_j}{\partial f_j} * \frac{\partial f_j}{\partial x_i} * inp_i,$$

де $\frac{\partial CEE}{\partial SM_j}$ – похідна похибки крос ентропії по відношенню до результату softmax для j-го виходу, $\frac{\partial SM_j}{\partial f_j}$ – похідна результату softmax для j-го виходу по відношенню до j-го входу в softmax (виходу j-ї функції активації), $\frac{\partial f_j}{\partial x_i}$ – похідна результату активаційної функції по відношенню до i-го входу активаційної функції, inp_i – значення i-го входу нейронної мережі.

Похідна похибки крос ентропії по відношенню до результату softmax виглядає наступним чином:

$$\frac{\partial CEE}{\partial SM_j} = -\sum_{j=1}^{n} \quad \frac{\partial (Target_j * log(Predicted_j))}{\partial Predicted_j} = -\sum_{j=1}^{n} \quad Target_j * \frac{1}{Predicted_j},$$

де *n* – кількість кількість виходів softmax.

Похідна j-го результату softmax по відношенню до результату i-ї активаційної функції згідно з [44] виглядає наступним чином:

$$\frac{\partial SM_j}{\partial f_i} = \{Predicted_j * (1 - Predicted_i), \text{якщо } i = j \; -Predicted_i * Predicted_j, \text{якщо } i \neq j$$

Зкомбінувавши $\frac{\partial CEE}{\partial SM_j}$ та $\frac{\partial SM_j}{\partial f_i}$ знаходимо $\frac{\partial CEE}{\partial f_i}$:

$$\frac{\partial CEE}{\partial f_i} = \{Predicted_i - 1, \text{якщо } i = j \; Predicted_i, \text{якщо } i \neq j$$

Далі треба визначити похідну активаційної функції. Оскільки в мережі була використана логістична активаційна функція, ми використаємо формулу (1), де $x = \sum_i^n \; w_{i,j} * inp_i$, множимо отримане значення на відповідний $\frac{\partial CEE}{\partial f_i}$ і після цього домножуємо на значення відповідного входу і отримуємо значення Δ. Далі множимо отримане значення Δ на крок навчання (step) – величину. яку задаємо власноруч для коригування швидкості та якості навчання і в фіналі модифікуємо ваги віднімаючи значення Δ * step від поточного значення ваги. Слід зазначити, що оскільки ми проводимо m кількість експерементів, то слід розрахувати Δ для кожної ваги в кожному експерименті окремо, а потім визначити середню Δ для кожної ваги:

$$\Delta_{i,j} = \frac{\sum_{k=1}^{m} \; (\Delta_{i,j})_k}{m},$$

де $(\Delta_{i,j})$ – зміна ваги (i, j), а m – кількість експериментів.

Узагальнена структура алгоритму навчання нейронної мережі представлена на рисунку 2.5

Нейронна мережа навчається поки помилка зменшується, коли вона починає збільшуватись, процес навчання припиняється.

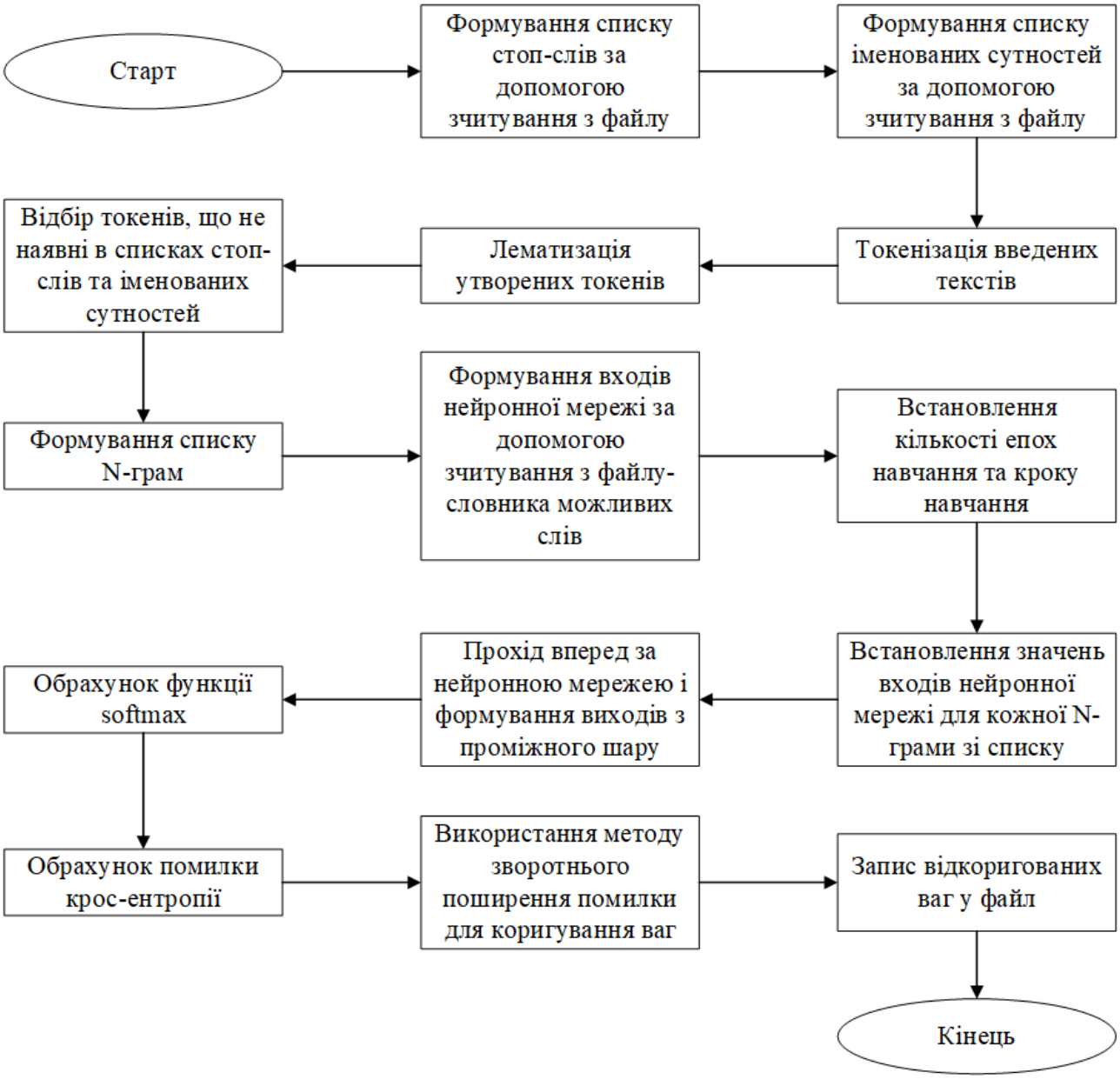

Рисунок 2.5 – Узагальнена структура алгоритму навчання нейронної мережі

Після того, як нейронна мережа була навчена, використовується тільки процес прямого поширення до шару softmax, і результати цього шару використовуються для формування списку найбільш ймовірних кандидатів в якості синонімів. Модулі, що виконують задачу перевірки на синонімію, використовують навчену нейронну мережу для перевірки відповідних слів в n-грамах на їх синонімічність шляхом завантаження в нейронну мережу вектору з

нулів та одиниць, де одиниці позначають контекст поточного слова і отримують в якості результату масив з значеннями ймовірностей використання різних слів в цьому контексті. У випадку, якщо було знайдено відповідне слово в перших елементах отриманого списку, маємо синонімію двох слів.

2.4.4 Алгоритм оцінювання

Після того, як був отриманий список n-грам для тексту, що перевіряється, з відповідними позначками, необхідно визначити оцінку роботи. Для універсальності будемо використовувати 100-бальну шкалу. В даному алгоритмі є два головні аспекти, на які треба звернути увагу: розмір відповіді і її відповідність еталону.

Для визначення коефіцієнту, що показує залежність оцінки від розмірів текстів, доцільно буде використати наступний підхід:

$$size\_coeficient = \{\frac{size(check)}{size(etalon)}, \text{якщо } size(etalon) > size(check) \frac{size(etalon)}{size(check)}, \text{якщо } size(etalon) < size(check),$$

де $size(etalon)$ – розмір списку n-грам еталону, а $size(check)$ – розмір списку n-грам тексту, що оцінюється.

Другою частиною є відповідність еталону. На основі різних типів подібності, отриманих в результаті порівняння на ідентичність та синонімічність, формується рівень повної подібності:

$$total\_similarity = \frac{\sum_{i=1}^{n} 1 * similarity\_level\_coeficient_i}{n},$$

де n - кількість n-грам в тексті, що перевіряється, а $similarity\_level\_coeficient$ – значення, що відповідає певному рівню подібності і задається користувачем. Для даної роботи будуть використані наступні $similarity\_level\_coeficient$:

$$similarity\_level\_coeficient_i =$$

$$\{1, \text{якщо буквально відповідає еталону } 0.95, \text{якщо синонімічно відповідає еталону } 0$$

Фінальним етапом є перемноження $total\_similarity$ і $size\_coeficient$ та домножування цього результату на 100. Це і буде оцінкою роботи.

2.5 Проектні рішення та вибір програмних засобів

Оскільки не було знайдено програмних засобів, які б вирішували поставлену проблему і було прийняте рішення про створення власного застосунку. Для цього необхідним є визначення мови програмування, її базового функціоналу, додаткових засобів для неї і середовища розробки. Мовою програмування для даного завдання було обрано Python, оскільки вона є дуже гнучкою, інтуїтивно зрозумілою та має багато вбудованого функціоналу, що допоможе при виконанні роботи. Щодо бібліотек, які використовуються в даному завданні, в якості додаткових засобів слід виділити наступні бібліотеки: tkinter, що надає великий набір інструментів для створення власного користувацького графічного інтерфейсу + ttkbootstrap для покращення вигляду інтерфейсу; PyPDF2 та python-docx для роботи з .pdf та .docx файлами відповідно; nltk для формування набору токенів та n-грам; lemmagen3 для лематизації отриманих токенів; spacy для вилучення іменованих сутностей; math та numpy для математичних операцій; os для роботи з системою (пошук файлів/папок в системі). Середовищем для розробки буде PyCharm. Це безкоштовне середовище, яке може запускати скрипти Python за допомогою різних інтерпретаторів а також

має зручну можливість відслідковування виконання коду з метою пошуку і виправлення помилок.

2.6 Область застосування та перспективність алгоритмічного підходу

Обраний алгоритмічний підхід має великий потенціал, щоб стати основою підходу для професійної системи оцінювання відповідей студентів на відкриті питання при проведені контрольних заходів в ЗВО, що є основною областю застосування, оскільки він охоплює і буквальний і синонімічний огляд, і потребує лише додаткових обчислюваних потужностей. Також його можна розширити для покращення результатів, що визначає його перспективність використання в більш складних системах.

Завдяки використанню n-грам у метода є можливість встановлювати певні зв’язки та послідовності, що є важливим елементом при оцінці відповідностей текстових даних.

Нейронна мережа, що застосовується має також достатньо великий потенціал для подальшого використання, оскільки навіть на невеликому розмірі даних, що не сильно розріджені, вона навчилась непогано встановлювати синонімію. А враховуючи те, що для неї можна застосувати значно більше даних для навчання та більшу кількість епох навчання за наявності відповідних обчислюваних потужностей, можна сказати, що вона має гарні шанси на ефективне виконання поставленої задачі.

Щодо алгоритму оцінювання, то він також задає базис для подальшої модифікації, але для цього необхідне залучення викладачів в якості експертів для корегування критеріїв оцінки.

2.7 Практична цінність розробки

Створення програмного застосунку, що автоматично перевіряє роботу студента має декілька практичних переваг. По-перше, це прискорення перевірки роботи і, відповідно, зняття навантаження з викладача, що підвищує ефективність роботи та здатність виконувати інші важливі активності По-друге, покращення навчального процесу завдяки тому, що студент не чекає довго на результат і має більше часу для виконання інших завдань. По-третє, це надасть змогу зробити процес перевірки робіт студентів більш об’єктивним. По-четверте, такий програмний засіб може бути в подальшому вдосконалений і стати частиною майбутньої цифрової системи освіти.

2.8 Висновки до другого розділу

Сучасні методи аналізу тексту використовуються в різних задачах обробки та генерації природномовної мови. Всі вони мають свої переваги та недоліки, що обумовлені задачами, які вони вирішують. Оскільки в даній роботі розв’язується доволі специфічна задача, то для її розв’язання треба підібрати правильний підхід.

В процесі аналізу різних підходів було прийняте рішення використати власний підхід на основі n-грам, які будуть використані в буквальному та синонімічному порівняннях текстів, що перевіряється, на встановлення загального рівня їх подібності. Для синонімічного порівняння було обрано метод, що використовує нейронну мережу та описаний принцип роботи мережі. Ця комбінація методів буде використана як для порівняння з еталоном, так і з текстами корпусу.

Також були розглянуті перспективність підходу та практична цінність застосунку, програмні рішення та засоби, що будуть використані. Розроблений підхід має потенціал для розвитку та професійного впровадження в майбутньому.

3 Програмна реалізація та тестовий приклад роботи системи

3.1 Опис інтерфейсу користувача

3.1.1 Головне меню - вікно «Застосунок»

Після запуску застосунку не екрані з’являється головне меню програми «Застосунок», яке містить 3 функціональні кнопки (рисунок 3.1).

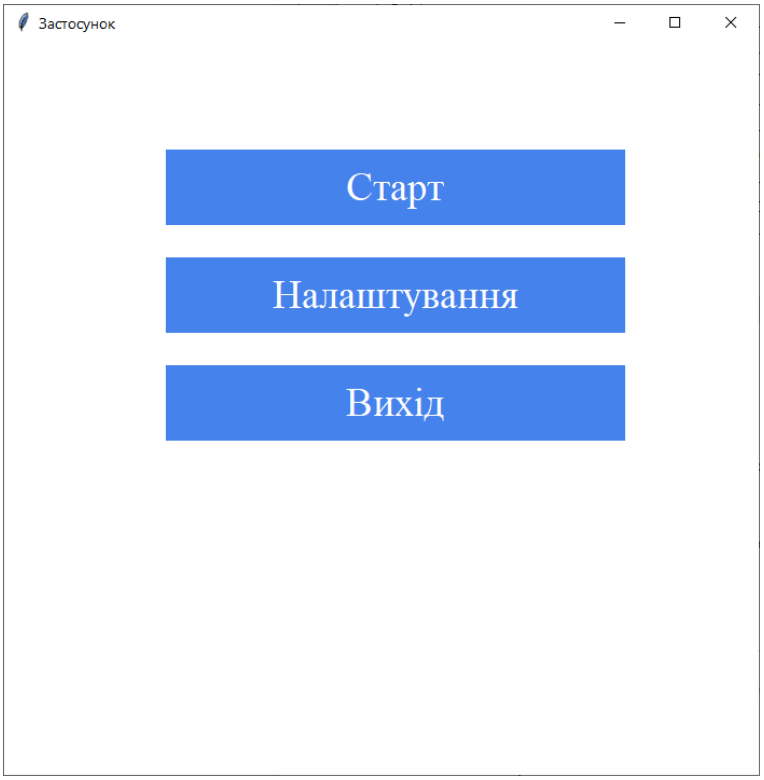

Рисунок 3.1 – Вікно «Застосунок»

Шляхом натискання на кнопки відбувається відкриття підпорядкованих вікон, або вихід із застосунку, а саме:

- кнопка «Старт» відкриває вікно «Вибір текстів», в якому наявний інтерфейс для додавання текстів, що порівнюються, та кнопки для запуску роботи алгоритму та закриття вікна;

- кнопка «Налаштування» відкриває однойменне вікно, в якому відображається перелік налаштувань системи, а також є поля для редагування відповідних налаштувань і кнопки для підтвердження нових налаштувань або закриття вікна;
- кнопка «Вихід» закриває вікно «Застосунок» та завершує роботу застосунку.

3.1.2 Вікно «Налаштування»

Вікно «Налаштування» (рисунок 3.2) є дочірнім для вікна «Застосунок».

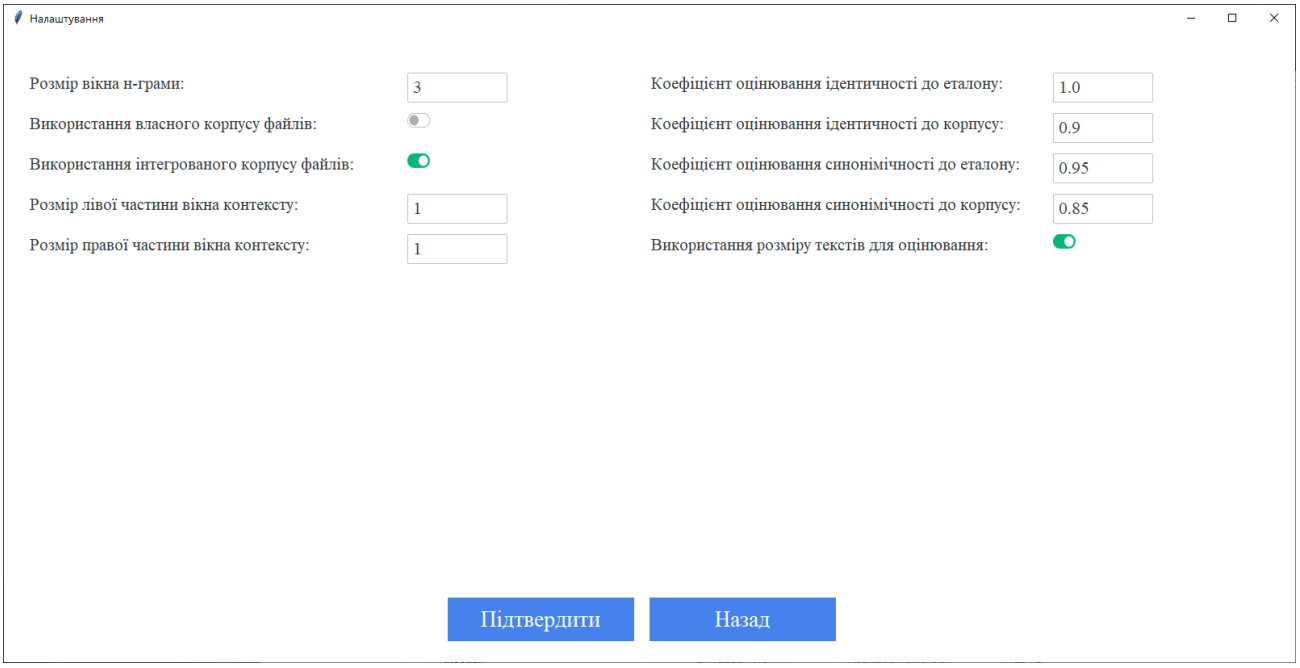

Рисунок 3.2 – Вікно «Налаштування»

У вікні відображаються налаштування для роботи системи, а саме:

- «Розмір вікна n-грами» – поле для визначення розміру вікна, що буде використовуватися при формуванні n-грам, за замовченням використовується значення «3»;

- «Використання власного корпусу файлів» – повзунок, який можна перемикнути, і таким чином відкрити вікно «Вибір корпусу», щоб обрати власний корпус файлів для порівняння, за замовченням використовується значення «не активний»;
- «Використання інтегрованого корпусу файлів» – повзунок, який можна перемикнути, щоб ввімкнути/вимкнути використання інтегрованого корпусу, за замовченням використовується значення «активний»;
- «Розмір лівої частини вікна контексту» – поле для визначення яка кількість токенів буде розглядатись як контекст слова ліворуч від нього, за замовченням використовується значення «1»;
- «Розмір правої частини вікна контексту» – поле для визначення яка кількість токенів буде розглядатись як контекст слова праворуч від нього, за замовченням використовується значення «1»;
- «Коефіцієнт оцінювання ідентичності до еталону» - поле для визначення коефіцієнта, що використовується в алгоритмі оцінювання, в тому випадку, якщо було знайдено ідентичну n-граму в еталоні;
- «Коефіцієнт оцінювання ідентичності до корпусу» - поле для визначення коефіцієнта, що використовується в алгоритмі оцінювання, в тому випадку, якщо було знайдено ідентичну n-граму в одному з текстів корпусу;
- «Коефіцієнт оцінювання синонімічності до еталону» - поле для визначення коефіцієнта, що використовується в алгоритмі оцінювання, в тому випадку, якщо було знайдено n-граму синонім в одному з текстів корпусу;
- «Коефіцієнт оцінювання синонімічності до корпусу» - поле для визначення коефіцієнта, що використовується в алгоритмі оцінювання, в тому випадку, якщо було знайдено ідентичну n-граму в одному з текстів корпусу;
- «Використання розміру текстів для оцінювання» - поле для визначення чи потрібно використовувати різницю в розмірах текстів для оцінювання;

Вікно містить дві кнопки:

- кнопка «Підтвердити» зберігає всі налаштування, наявні у вікні та закриває вікно «Налаштування»;
- кнопка «Назад» закриває вікно «Налаштування» без збереження налаштувань;

3.1.3 Вікно «Вибір корпусу»

Вікно «Вибір корпусу» (рисунок 3.3) є дочірнім для вікна «Налаштування».

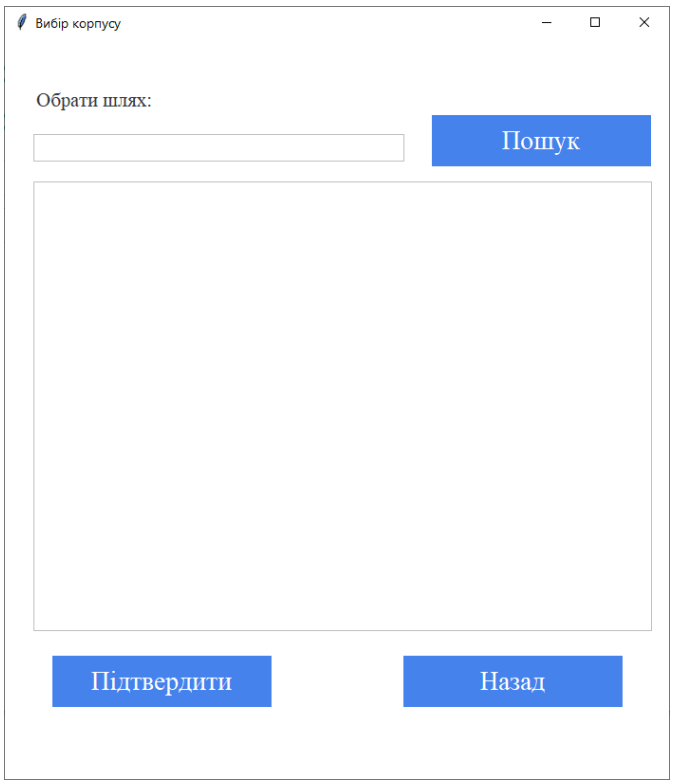

Рисунок 3.3 – Вікно «Вибір корпусу»

За допомогою цього вікна здійснюється вибір папки з файлами, що буде власним корпусом. Вікно має наступні елементи:

- «Обрати шлях» – поле для відображення обраного шляху до папки, що є корпусом;

- текстове поле під полем «Обрати шлях» – поле в якому відображаються файли, що наявні в папці і будуть завантажені в якості текстів корпусу;

Вікно містить три кнопки:

- кнопка «Пошук» відкриває діалогове вікно для вибору папки в системі, що буде використана як корпус;
- кнопка «Підтвердити» зберігає всі налаштування, наявні у вікні «Вибір корпусу» та закриває вікно «Вибір корпусу»;
- кнопка «Назад» закриває вікно «Налаштування» без збереження налаштувань;

3.1.4 Вікно «Вибір текстів»

Вікно «Вибір текстів» (рисунок 3.4) є дочірнім для вікна «Застосунок».

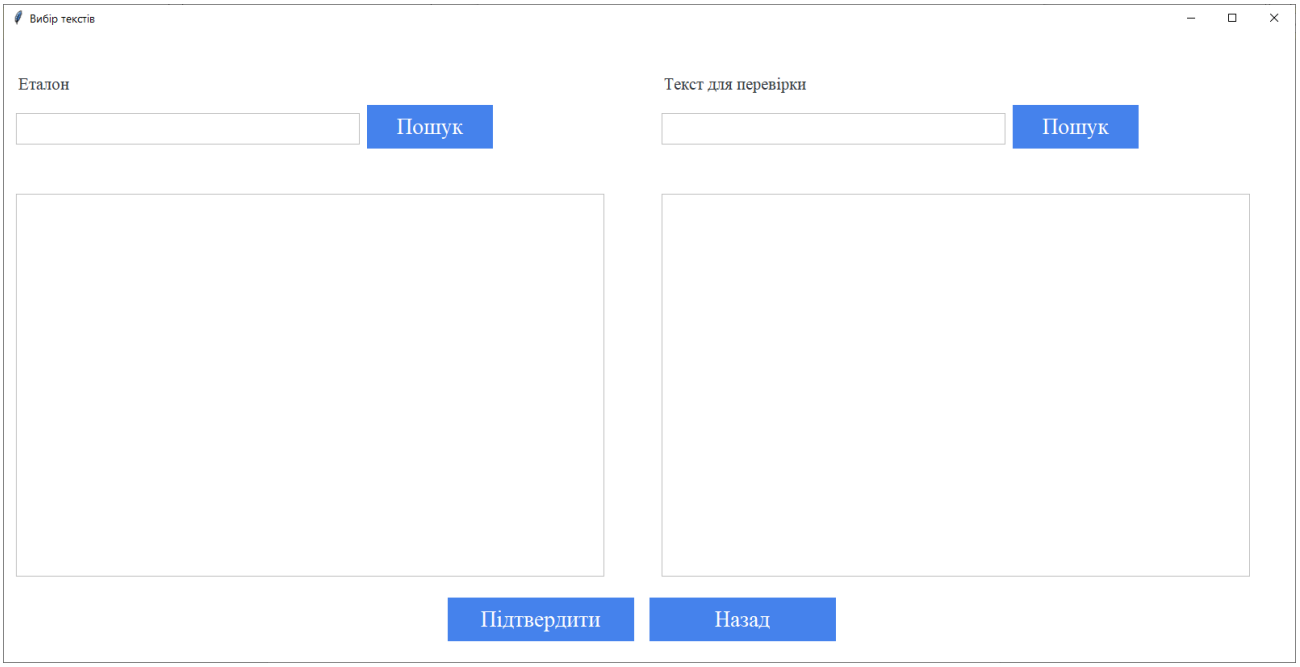

Рисунок 3.4 – Вікно «Вибір текстів»

За допомогою цього вікна обираються еталонний варіант відповіді та відповідь, що порівнюється. Вікно має наступні елементи:

- «Еталон» – поле для відображення обраного шляху до файлу, що є еталоном;
- Текстове поле під полем «Еталон» - поле для відображення тексту, наявного в файлі еталону;
- «Текст для перевірки» – поле для відображення обраного шляху до файлу, що є текстом для перевірки;
- Текстове поле під під полем «Текст для перевірки» - поле для відображення тексту, наявного в файлі для перевірки;

Вікно містить чотири кнопки:

- кнопки «Пошук» відкривають діалогові вікна для вибору текстового файлу в системі, що буде використана як еталон або файл для перевірики відповідно до поля біля якого розташована кнопка;
- кнопка «Підтвердити» зчитує завантажену інформацію та всі налаштування і запускає роботу алгоритму;
- кнопка «Назад» закриває вікно.

3.1.5 Вікно «Результат перевірки»

Вікно «Результат перевірки» (рисунок 3.5) є дочірнім для вікна «Вибір текстів».

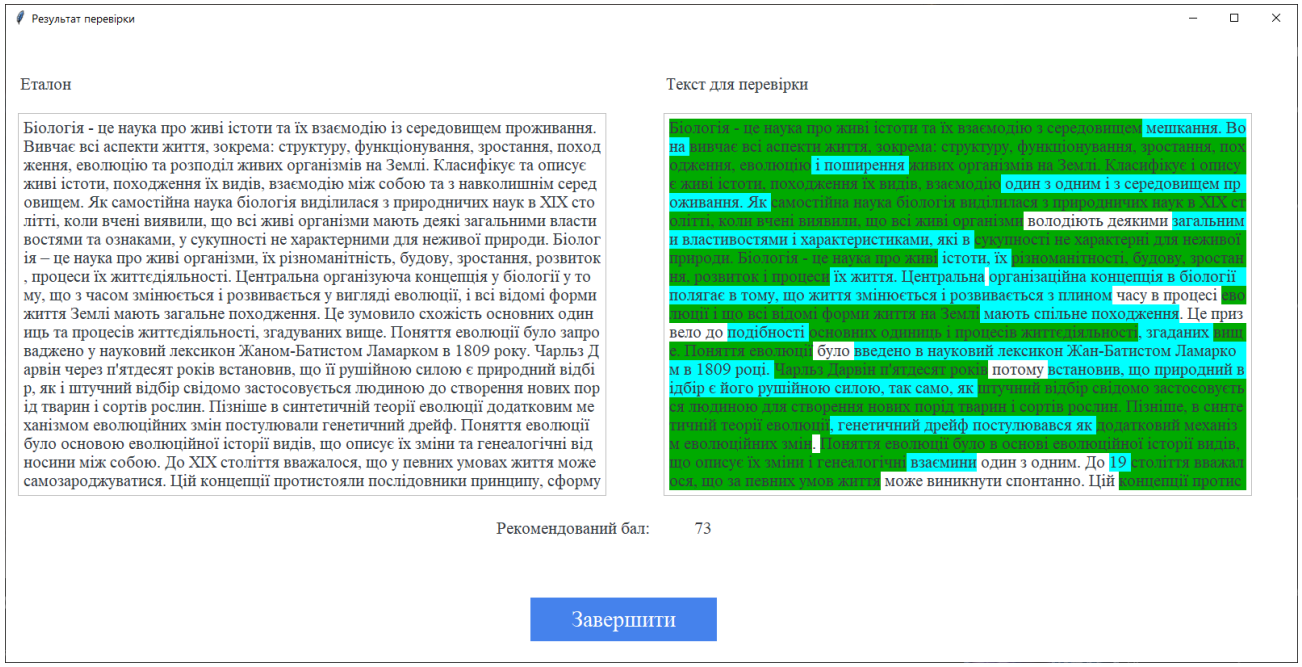

Рисунок 3.5 – Вікно «Результат перевірки»

Вікно результатів перевірки має наступні елементи:

- «Еталон» – поле для відображення тексту, наявного в файлі еталону;
- «Текст для перевірки» – поле для відображення тексту, наявного в файлі для перевірки з позначенням рівнів подібності;
- «Рекомендований бал» – запис, що відображає рекомендований бал за роботу;

Вікно містить одну кнопку:

- кнопка «Завершити» закриває вікно.

3.2 Тестові приклади роботи програмного застосунку

Приклад роботи програмного застосунку із застосуванням власного корпусу, з розміром вікна для n-грам рівним 3, з розмірами вікна контексту рівними 1, з коефіцієнтом оцінювання ідентичності до еталону рівним 1, з коефіцієнтом

оцінювання синонімічності до еталону рівним 0.95, з коефіцієнтом оцінювання синонімічності до корпусу рівним 0.85 для обраних еталону та тексту, що перевіряється, відображено на рисунках 3.6 та 3.7.

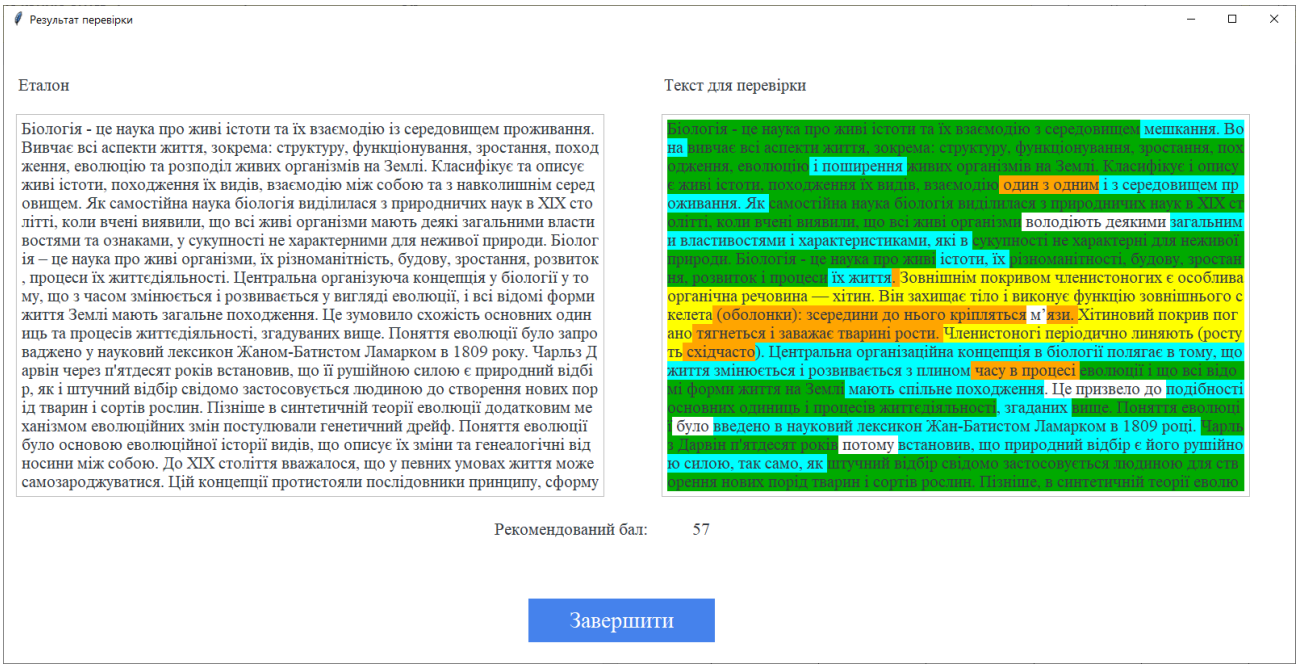

Рисунок 3.6 – Вікно «Приклад роботи програмного застосунку ч.1»

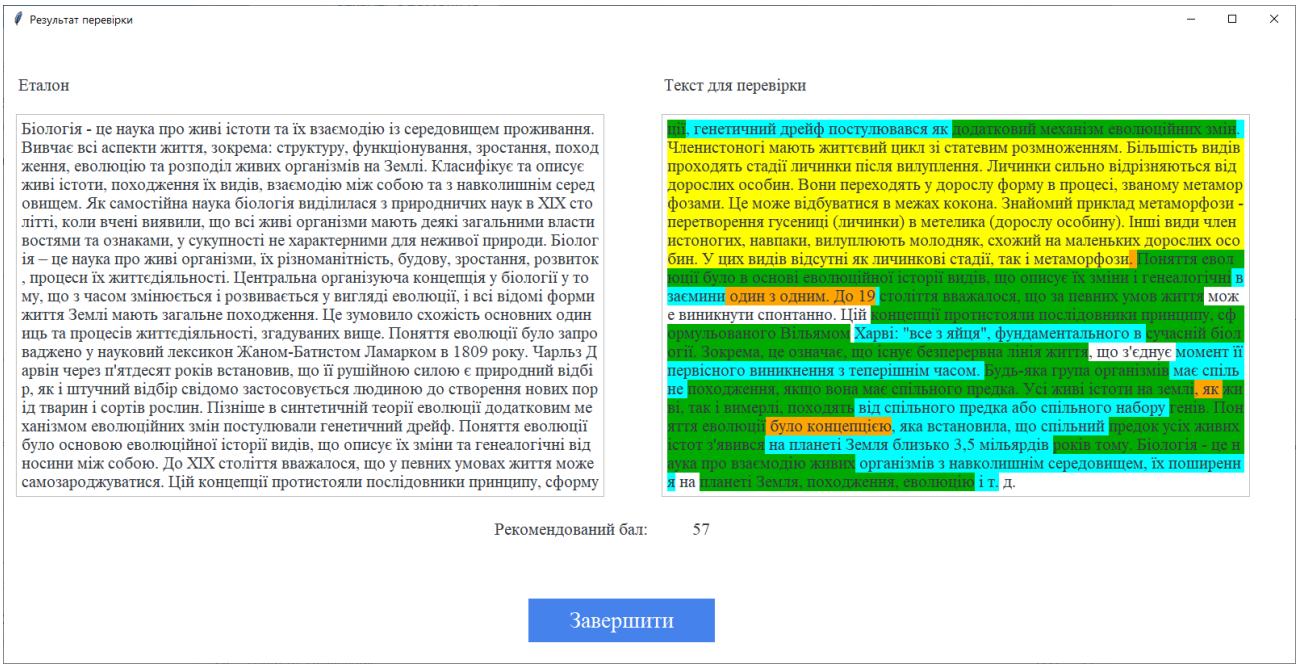

Рисунок 3.7 – Вікно «Приклад роботи програмного застосунку ч.2»

На представлених зображеннях результатів можна побачити виділення різного кольору для тексту, що перевірявся (у відповідності до кольорів, описаних в другому розділі) і рекомендований бал роботи, що і було метою роботи.

Також були проведені експерименти з різними параметрами алгоритму оцінювання з використанням текстів корпусу. Результати для цих експериментів наведено в таблиці 3.2:

Таблиця 3.2 Експерименти з різними параметрами алгоритму оцінювання з використанням текстів корпусу.

Використання розміру текстів для оцінювання	Коефіцієнт оцінювання ідентичності до еталону	Коефіцієнт оцінювання ідентичності до корпусу	Коефіцієнт оцінювання синонімічності до еталону	Коефіцієнт оцінювання синонімічності до корпусу	Рекомендований бал
Так	1.0	0.9	0.95	0.85	57
Так	1.0	1.0	1.0	1.0	60
Ні	1.0	0.9	0.95	0.85	81
Ні	1.0	1.0	1.0	1.0	85

3.3 Інструктивні матеріали користувача

Для початку експлуатації необхідно виконати наступний інсталяційний процес:

Крок 1. Запустити файл Setup.exe.

Крок 2. Обрати мову установки (рисунок 3.8).

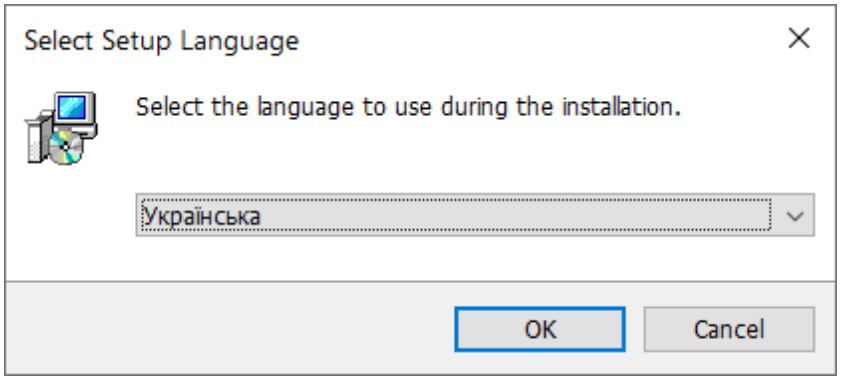

Рисунок 3.8 – Вибір мови установки

Крок 3. Вибір шляху встановлення програми (рисунок 3.9).

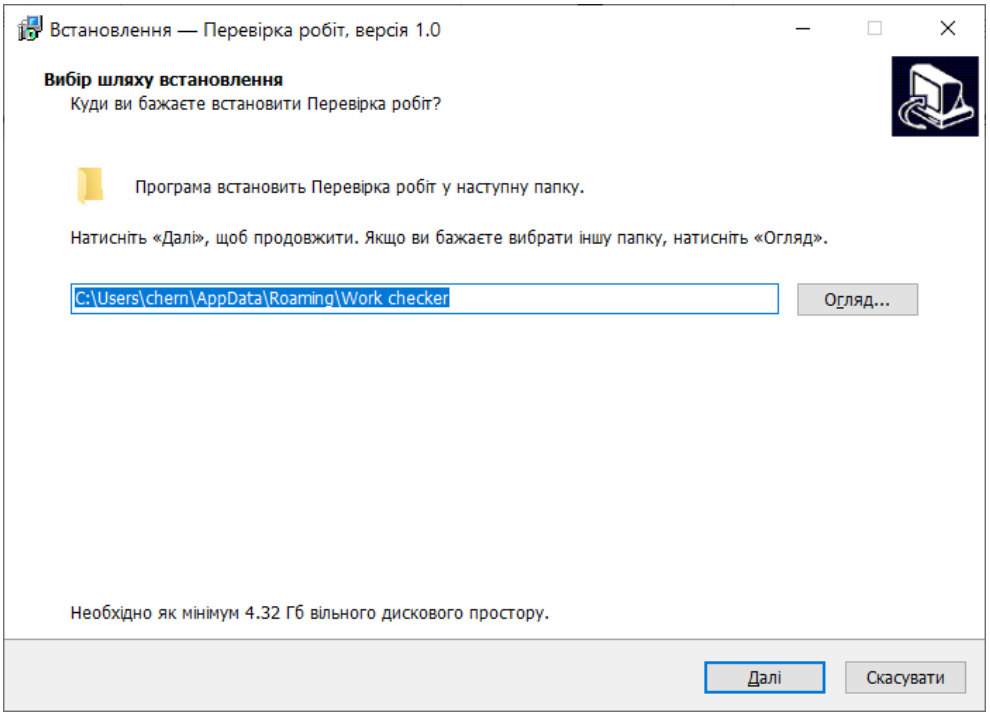

Рисунок 3.9 – Вибір шляху встановлення програми

Крок 4. Вибір назви папки в меню «Пуск» (рисунок 3.10).

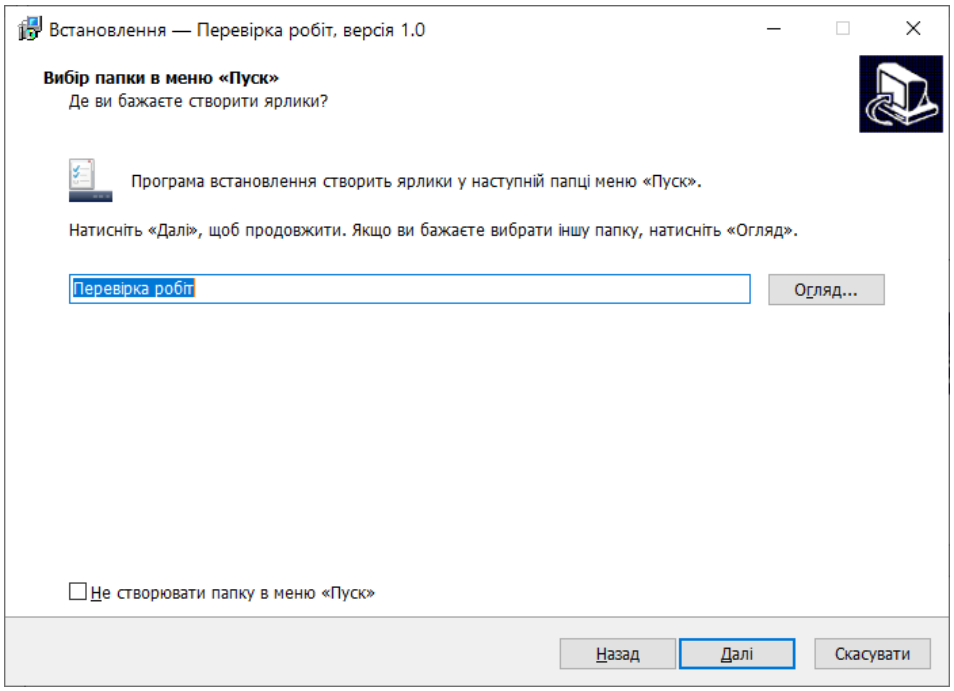

Рисунок 3.10 – Вибір назви папки в меню «Пуск»

Крок 5. Вибір додаткових завдань (рисунок 3.11).

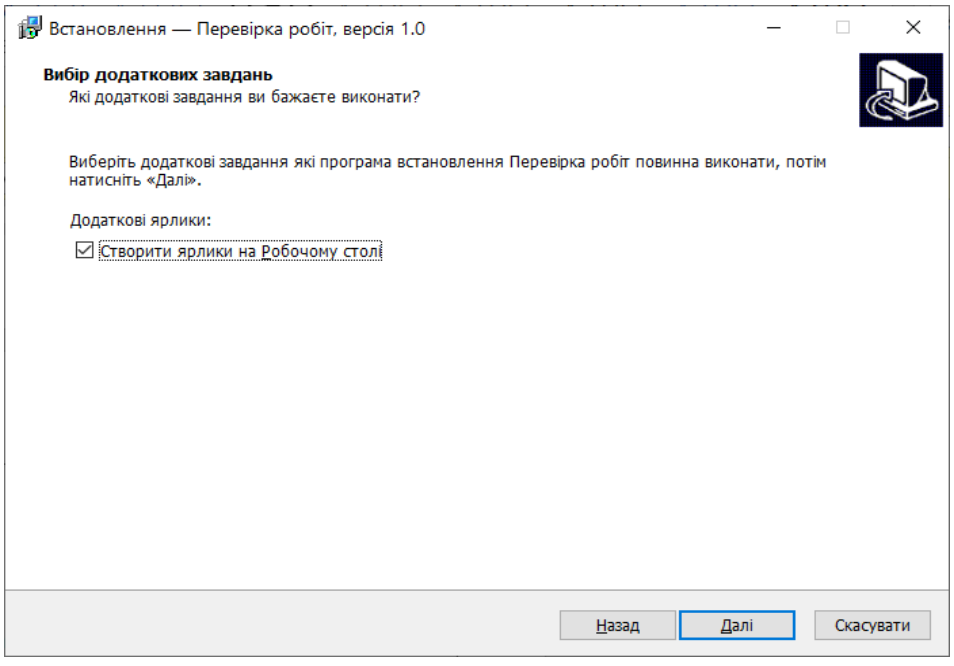

Рисунок 3.11 – Вибір додаткових завдань

Крок 6. Перевірка налаштувань встановлення програми (рисунок 3.12).

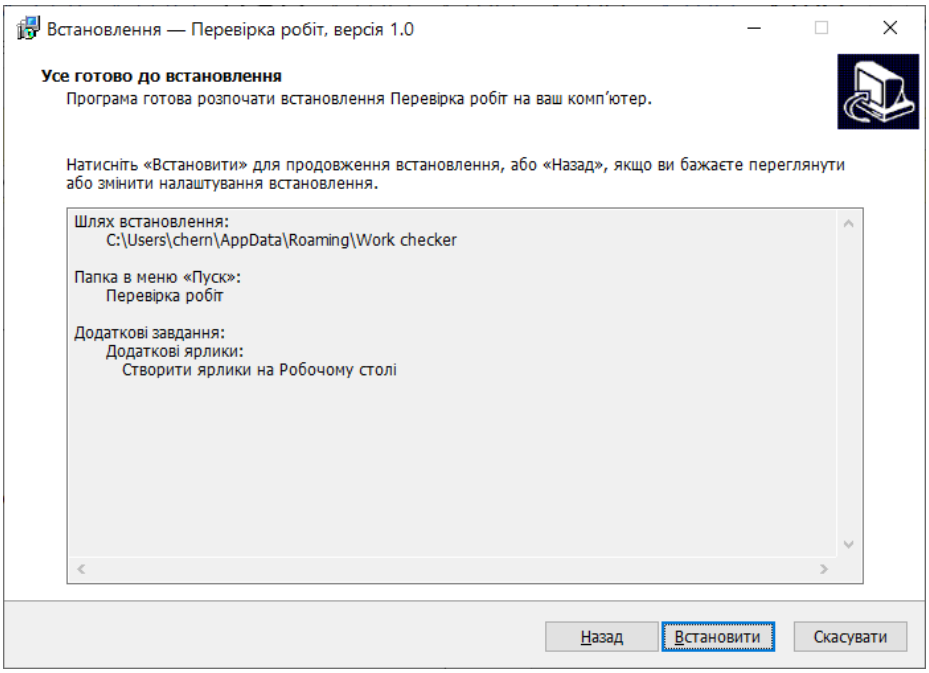

Рисунок 3.12 – Перевірка налаштувань встановлення програми

Крок 7. Прогрес встановлення програми (рисунок 3.13).

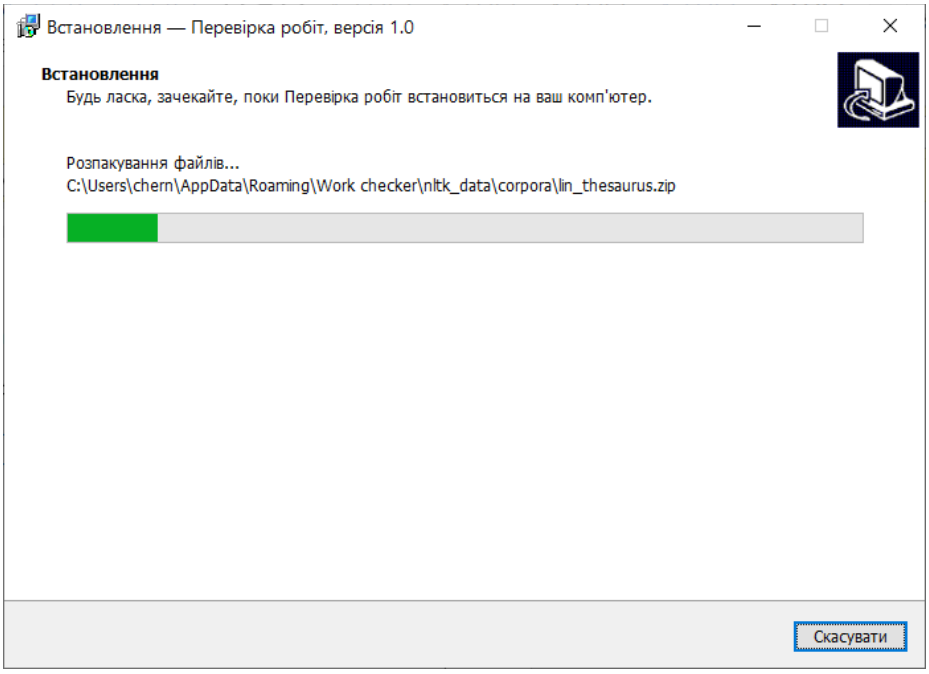

Рисунок 3.13 – Прогрес встановлення програми

Крок 8. Завершення встановлення програми (рисунок 3.14).

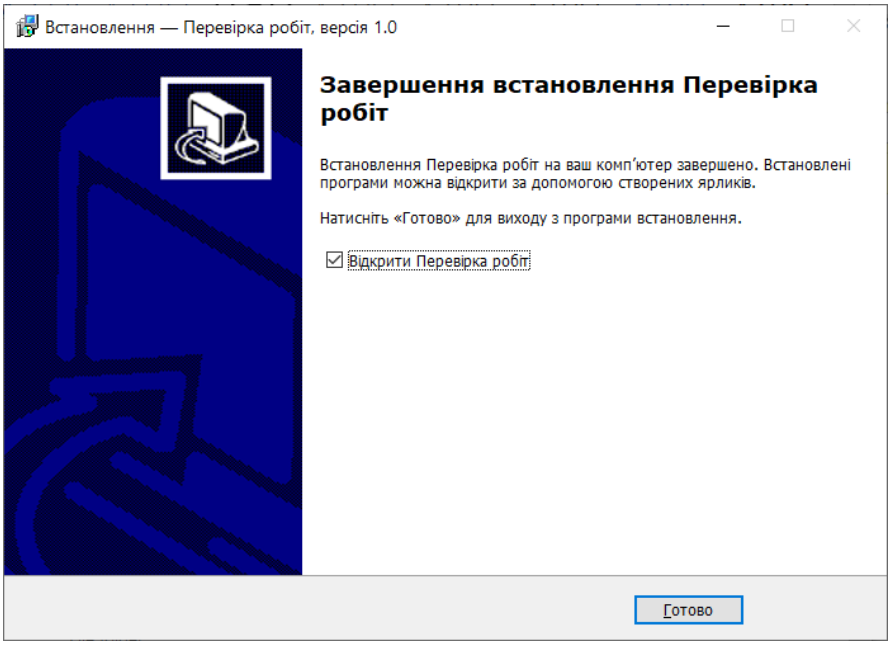

Рисунок 3.14 – Завершення встановлення програми

Після запуску програмного забезпечення, користувач може настиснути кнопку «Старт» для початку роботи, в такому випадку використовуються налаштування за замовчуванням, або кнопку «Налаштування» для встановлення власних налаштувань.

При визначенні власного корпусу у налаштуваннях необхідно обирати папку, що складається з файлів з розширенням .docx або .pdf.

Після вибору власного корпусу необхідно натиснути кнопку «Підтвердити» у вікні «Вибір корпусу», щоб зберегти вибір.

Після встановлення всіх налаштувань необхідно натиснути кнопку «Підтвердити» у вікні «Налаштування», щоб зберегти вибір.

Після того, як користувач натиснув кнопку «Старт» в вікні головного меню, відкривається вікно «Вибір текстів», в якому користувач повинен вибрати два тексти для порівняння. Для вибору файлів в системі необхідно натиснути одну з кнопок «Пошук»: перша (ліва) відповідає за вибір еталону, друга (права) – за текст, що перевіряється. Файли повинні бути в розширенні .docx або .pdf. Після

цього необхідно натиснути кнопку «Підтвердити», щоб запустити роботу алгоритму.

Тепер необхідно почекати деякий час поки працює алгоритм, і через деякий час буде отримано результат порівняння в вигляді тексту з позначками та оцінки.

3.4 Висновки до третього розділу

Створений програмний застосунок дозволяє порівняти між собою еталон/корпус та відповідь, що перевіряється і, визначити рівень відповідності даної відповіді до бажаної, і на основі цього встановити рекомендований бал. Також система має зрозумілий лаконічний користувацький інтерфейс та надає функціонал для налаштування.

В процесі тестування програмного застосунку було встановлено, що нейронна мережа навчається доволі гарно, результати роботи застосунку дозволяють об’єктивно оцінювати відповіді студентів. Хоча для покращення результатів треба збільшити обсяг навчальних даних для нейронної мережі і обрати необхідні крок та кількість епох. Але слід зазначити, що навчання мережі та перевірка на синонімію потребує великих ресурсів і якщо збільшувати обсяги навчання, або кількість шарів мережі, треба збільшити й апаратні можливості.

В цілому система має потенціал і при деяких модифікаціях і розвитку може бути використана в ЗВО України.

ВИСНОВКИ

Отже, підводячи підсумки можна зробити наступні висновки:

1. Розглянута проблема є актуальною і такою, що потребує вирішення, оскільки програмні засоби, що наявні на ринку, орієнтовані на інші, більш загальні проблеми.

2. На поточний момент в ЗВО використовуються програмні засоби для проведення та оцінювання тестів, але для перевірки відповідей на відкриті питання таких засобів немає, тому запровадження відповідного засобу може бути доволі актуальним.

3. Функціонал програми допомагає, в деякій мірі, автоматизувати процес перевірки робіт студентів і заощадити час та людські сили викладача на перевірку.

4. Розробка програмного застосунку допомагає продемонструвати процес поєднання буквального та синонімічного порівняння текстів із застосуванням n-грам та застосування алгоритму оцінювання.

5. Впровадження програмного застосунку дозволить автоматизувати процес проведення контрольних заходів, зменшити навантаження на викладачів та позбутися помилок при прийнятті рішень шляхом зменшення впливу людського фактору.

Виконання даної роботи дозволяє продемонструвати знання та навички, здобуті в процесі навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nataliia Tmienova, Arthur Chernenko - Approach to Automatic Verification of Answers to Open Questions When Performing Control Measures, IX International conference IT&I-2022 - Taras Shevchenko National University of Kyiv - - Kyiv – Ukraine - 31.11.2022 - 02.12.2022, pp. 156 – 158.

2. Karin Verspoor, Kevin Bretonnel Cohen – Natural Language Processing – Encyclopedia of Systems Biology – January 2013 – pp. 1495-1498 – DOI: 10.1007/978-1-4419-9863-7_158 - https://www.researchgate.net/publication/291179558_Natural_Language_Processing

3. Diksha Khurana, Aditya Koli, Kiran Khatter and Sukhdev Singh – Natural Language Processing: State of The Art, Current Trends and Challenges - Multimedia Tools and Applications - 14 July 2022 – DOI: 10.1007/s11042-022-13428-4 - https://arxiv.org/ftp/arxiv/papers/1708/1708.05148.pdf

4. Ximena Bolanos - Natural Language Processing with Machine Learning - https://www.encora.com/insights/natural-language-processing-with-machine-learning

5. The Pros and Cons of Using Natural Language Processing Tools - https://www.wesuggestsoftware.com/the-pros-and-cons-of-using-natural-language-processing-tools

6. Chetna Khanna - Text pre-processing: Stop words removal using different libraries - Towards Data Science - 10 February 2021 - https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a

7. Text Cleaning for NLP: A Tutorial - https://monkeylearn.com/blog/text-cleaning

8. When (not) to Lemmatize or Remove Stop Words in Text Preprocessing - https://www.modelop.com/blog/when-not-to-lemmatize-or-remove-stop-words-in-text-preprocessing

9. Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, Denny Zhou - Fast WordPiece Tokenization - Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing - Punta Cana - Dominican Republic - 7–11 November 2021 – pp. 2089–2103 - https://aclanthology.org/2021.emnlp-main.160.pdf

10. Dr. S. Vijayarani, Ms. R. Janani - Text Mining: Open Source Tokenization Tools – An Analysis - An International Journal - No.1 - January 2016 - https://www.researchgate.net/publication/329800669_Text_Mining_Open_Source_Tokenization_Tools_An_Analysis

11. Oana Frunza - A Trainable Tokenizer, solution for multilingual texts and compound expression tokenization - Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08) - European Language Resources Association (ELRA) – Marrakech - Morocco – May 2008 - http://www.lrec-conf.org/proceedings/lrec2008/pdf/152_paper.pdf

12. Ms. Anjali Ganesh Jivani - A Comparative Study of Stemming Algorithms - International Journal of Computer Technology and Applications - 2011 - https://kenbenoit.net/assets/courses/tcd2014qta/readings/Jivani_ijcta2011020632.pdf

13. Divya Khyani, Siddhartha B S, Niveditha N M,Divya B M - An Interpretation of Lemmatization and Stemming in Natural Language Processing - Journal of University of Shanghai for Science and Technology - January 2021 - https://www.researchgate.net/publication/348306833_An_Interpretation_of_Lemmatization_and_Stemming_in_Natural_Language_Processing

14. Enrique Manjavacas, Akos Kadar, Mike Kestemont - Improving Lemmatization of Non-Standard Languages with Joint Learning - Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Association for Computational Linguistics – Minneapolis - Minnesota - June 2019 - Volume 1 – pp. 1493-1503 - https://arxiv.org/pdf/1903.06939.pdf

15. Afham Fardeen - Tutorial on Spacy Part of Speech (POS) Tagging - https://machinelearningknowledge.ai/tutorial-on-spacy-part-of-speech-pos-tagging

16. Tiago Duque - Building a Part of Speech Tagger - Analytics Vidhya – 21 Februaty 2020 https://medium.com/analytics-vidhya/part-of-speech-tagging-what-when-why-and-how-9d250e634df6

17. Daniel Jurafsky & James H. Martin - Speech and Language Processing - Prentice Hall – 16 May 2008 - https://web.stanford.edu/~jurafsky/slp3/3.pdf

18. Natural language processing. N-gram language model and its application - https://www.programmersought.com/article/53343829420

19. Sauleh Eetemadi, William Lewis, Kristina Toutanova, Hayder Radha - Survey of Data-Selection Methods in Statistical Machine Translation - Machine Translation Journal - December 2015 – DOI: 10.1007/s10590-015-9176-1 - https://www.microsoft.com/en-us/research/wp-content/uploads/2016/04/MTJ-final-Editor-v1.pdf

20. Ye Zhang, Byron C. Wallace - A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification - Proceedings of the Eighth International Joint Conference on Natural Language Processing - Asian Federation of Natural Language Processing – Taipei - Taiwan - November 2017 - Volume 1 – pp. 253–263 - https://aclanthology.org/I17-1026.pdf

21. Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Galle, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, Samson Tan - Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP - Computing Research Repository – 20 December 2021 - https://arxiv.org/pdf/2112.10508.pdf

22. Anudhyan Boral, Sylvain Schmitz - Model Checking Parse Trees - LICS'13: Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science - June 2013 – pp. 153–162 - https://arxiv.org/pdf/1211.5256.pdf

23. Boris Galitsky - Learning Parse Structure of Paragraphs and its Applications in Search - Engineering Applications of Artificial Intelligence 26(3) - March 2013 – pp. 1072–1091 - https://www.researchgate.net/publication/257392770_Machine_learning_of_syntactic_parse_trees_for_search_and_classification_of_text

24. Madhuri A. Tayal, Dr. M. M. Raghuwanshi, Dr. Latesh Malik - Syntax Parsing: Implementation using Grammar-Rules for English Language - 11th International Conference on Natural Language Processing - Goa University - Goa – India - 18-21 December 2014 - pp. 139-145 - https://www.researchgate.net/publication/260034501_Syntax_Parsing_Implementation_using_Grammar-Rules_for_English_Language

25. Recursive Descent Parser - https://www.geeksforgeeks.org/recursive-descent-parser

26. Irvi Aini - Recursive Descent Parser - https://irvifa.medium.com/recursive-descent-parser-573641b461ed

27. Miloš Jakubícek - Rule-Based Parsing of Morphologically Rich Languages - Masaryk University Faculty of Informatics – Brno - 2017 - https://nlp.fi.muni.cz/~xjakub/phd/dis-draft.pdf

28. Faisal Rahutomo, Teruaki Kitasuka, Masayoshi Aritsugi - Semantic Cosine Similarity - The 7th International Student Conference on Advanced Science and Technology ICAST 2012 - Seoul - South Korea - https://www.researchgate.net/publication/262525676_Semantic_Cosine_Similarity

29. N. Adilah Hanin Zahri, Fumiyo Fukumoto, Matsyoshi Suguru, Ong Bi Lynn – Application of Rhetorical Relations Between Sentences to Clusters-Based Text Summarization - Fifth International Conference on Computer Science, Engineering and Applications (CCSEA-2015) – Dubai - UAE – 23-24 January 2015 - https://airccj.org/CSCP/vol5/csit53307.pdf

30. Delphine Charlet and Geraldine Damnati - SimBow at SemEval-2017 Task 3: Soft-Cosine Semantic Similarity between Questions for Community Question Answering - Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017) - Association for Computational Linguistics – Vancouver - Canada – August 2017 – pp. 315–319 - https://aclanthology.org/S17-2051.pdf

31. Sheikh Abujar, Mahmudul Hasan, Syed AkhterHossain - Sentence Similarity Estimation for Text Summarization Using Deep Learning - 2nd International Conference on Data Engineering and Communication Technology (ICDECT) - Pune – India - 2017 - Volume: Advances in Intelligent Systems and Computing (AISC) Series of Springer - https://www.researchgate.net/publication/321170679_Sentence_Similarity_Estimation_for_Text_Summarization_using_Deep_Learning

32. Rishin Haldar and Debajyoti Mukhopadhyay - Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach - https://arxiv.org/ftp/arxiv/papers/1101/1101.1232.pdf

33. Shahzad Qaiser, Ramsha Ali - Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents - International Journal of Computer Applications – No.1, July 2018 - https://www.researchgate.net/publication/326425709_Text_Mining_Use_of_TF-IDF_to_Examine_the_Relevance_of_Words_to_Documents

34. https://plagiarismcheckerx.com/?AFFILIATE=120043&__c=1

35. https://www.copyscape.com/about.php

36. https://windowsreport.com/plagiarism-software

37. https://unicheck.com/uk-ua

38. https://www.scribbr.com/plagiarism/best-free-plagiarism-checker

39. Murad Abu-Khalaf – EE 5322 Neural Networks Notes - https://lewisgroup.uta.edu/ee5322/lectures/NeuralNets.pdf

40. R. Rojas - Neural Networks - Springer-Verlag - Berlin - 1996 - https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf

41. Kiprono Elijah Koech - Derivative of Sigmoid and Cross-Entropy Functions - Towards Data Science – 10 August 2022 - https://towardsdatascience.com/derivative-of-sigmoid-and-cross-entropy-functions-5169525e6705

42. J.G. Makin – Backpropagation - https://www.cs.cornell.edu/courses/cs5740/2016sp/resources/backprop.pdf

43. https://www.pnw.edu/wp-content/uploads/2020/03/Lecture-Notes-8-1-2.pdf

44. Classification and Loss Evaluation - Softmax and Cross Entropy Loss - https://deepnotes.io/softmax-crossentropy

Додаток А

Код модулю користувацького інтерфейсу

```
from tkinter import Tk, Frame, IntVar, Button, Toplevel, Label, StringVar, Entry, Text, END, HORIZONTAL
from tkinter import ttk
from ttkbootstrap import Style, Checkbutton
from ttkbootstrap.constants import *
from tkinter.filedialog import askopenfilename, askdirectory
from os import listdir
from os.path import isfile, join
import PyPDF2
import re
from docx import Document
import literal_etalon_copy
import etalon_synonymus
import literal_personal_corpus_copy
import personal_corpus_synonymus
import assessment

class MainWindow:
    def __init__(self, master):
        master.title("Застосунок")
        master.geometry('700x680')
        self.frame = Frame(master, width = 700, height = 680)
        self.settings = Settings()
        myStyle = Style()
        button_1 = Button(self.frame, text="Старт", font = "Times 24", width = 20, command=self.start)
        button_1.place(x=150, y=100)
        button_2 = Button(self.frame, text="Налаштування", font = "Times 24", width = 20, command=self.options)
        button_2.place(x=150, y=200)
        button_3 = Button(self.frame, text="Вихід", font = "Times 24", width = 20, command=master.destroy)
        button_3.place(x=150, y=300)
        self.frame.pack()

    def start(self):
        input_window = InputWindow(self.defineSettings())

    def options(self):
```

```
        settings_window = SettingsWindow(self.updateSettings,
                                         self.defineSettings())

    def updateSettings(self,
                       n_gram_count,
                       is_personal_corpus,
                       personal_corpus,
                       is_integrated_corpus,
                       network_window_left,
                       network_window_right,
                       etalon_literal_coeficient,
                       corpus_literal_coeficient,
                       etalon_synonymus_coeficient,
                       corpus_synonymus_coeficient,
                       use_size_coeficient):
        self.settings.setSettings(n_gram_count,
                                  is_personal_corpus,
                                  personal_corpus,
                                  is_integrated_corpus,
                                  network_window_left,
                                  network_window_right,
                                  etalon_literal_coeficient,
                                  corpus_literal_coeficient,
                                  etalon_synonymus_coeficient,
                                  corpus_synonymus_coeficient,
                                  use_size_coeficient)

    def defineSettings(self):
        return self.settings.getSettings()

class Settings:
    def __init__(self):
        self.n_gram_count = IntVar()
        self.n_gram_count = 3
        self.is_personal_corpus = IntVar()
        self.is_personal_corpus = 0
        self.personal_corpus = PersonalCorpus()
        self.is_integrated_corpus = IntVar()
        self.is_integrated_corpus = 1
```

```
        self.network_window_left = IntVar()
        self.network_window_left = 1
        self.network_window_right = IntVar()
        self.network_window_right = 1
        self.etalon_literal_coeficient = StringVar()
        self.etalon_literal_coeficient = "1.0"
        self.corpus_literal_coeficient = StringVar()
        self.corpus_literal_coeficient = "0.9"
        self.etalon_synonymus_coeficient = StringVar()
        self.etalon_synonymus_coeficient = "0.95"
        self.corpus_synonymus_coeficient = StringVar()
        self.corpus_synonymus_coeficient = "0.85"
        self.use_size_coeficient = IntVar()
        self.use_size_coeficient = 1

    def getSettings(self):
        return self.n_gram_count,\
               self.is_personal_corpus,\
               self.personal_corpus,\
               self.is_integrated_corpus,\
               self.network_window_left,\
               self.network_window_right,\
               self.etalon_literal_coeficient,\
               self.corpus_literal_coeficient,\
               self.etalon_synonymus_coeficient,\
               self.corpus_synonymus_coeficient,\
               self.use_size_coeficient

    def setSettings(self, n_gram_count, is_personal_corpus, personal_corpus, is_integrated_corpus,
network_window_left, network_window_right,
                    etalon_literal_coeficient, corpus_literal_coeficient, etalon_synonymus_coeficient,
corpus_synonymus_coeficient, use_size_coeficient):
        self.n_gram_count = n_gram_count
        self.is_personal_corpus = is_personal_corpus
        self.personal_corpus = personal_corpus
        self.is_integrated_corpus = is_integrated_corpus
        self.network_window_left = network_window_left
        self.network_window_right = network_window_right
        self.etalon_literal_coeficient = etalon_literal_coeficient
```

```
        self.corpus_literal_coeficient = corpus_literal_coeficient
        self.etalon_synonymus_coeficient = etalon_synonymus_coeficient
        self.corpus_synonymus_coeficient = corpus_synonymus_coeficient
        self.use_size_coeficient = use_size_coeficient

class SettingsWindow:
    def __init__(self, updateSettings, defineSettings):
        self.top = Toplevel()
        self.top.title("Налаштування")
        self.top.focus_set()
        self.frame = ttk.Frame(self.top, width = 1600, height = 780)
        self.updateSettings = updateSettings
        self.defineSettings = defineSettings
        self.label_1 = ttk.Label(self.top, text = "Розмір вікна н-грами: ", font = "Times 14")
        self.label_1.place(x=30, y=50)
        self.n_gram_count = IntVar(value=self.defineSettings[0])
        self.entry_1 = ttk.Entry(self.top, width=10, font = "Times 14", textvariable = self.n_gram_count)
        self.entry_1.place(x=500, y=50)
        self.label_2 = ttk.Label(self.top, text = "Використання власного корпусу файлів: ", font = "Times 14")
        self.label_2.place(x=30, y=100)
        self.is_personal_corpus = IntVar(value=self.defineSettings[1])
        self.personal_corpus = PersonalCorpus()
        self.check_1 = Checkbutton(self.top, bootstyle=SUCCESS + "-round-toggle", variable=self.is_personal_corpus,
onvalue=1, offvalue=0, command=self.choosePersonalCorpus)
        self.check_1.place(x=500, y=100)
        self.label_3 = ttk.Label(self.top, text="Використання інтегрованого корпусу файлів: ", font="Times 14")
        self.label_3.place(x=30, y=150)
        self.is_integrated_corpus = IntVar(value=self.defineSettings[3])
        self.check_2 = Checkbutton(self.top, bootstyle=SUCCESS + "-round-toggle", variable=self.is_integrated_corpus,
onvalue=1, offvalue=0)
        self.check_2.place(x=500, y=150)
        self.label_4 = ttk.Label(self.top, text="Розмір лівої частини вікна контексту: ", font="Times 14")
        self.label_4.place(x=30, y=200)
        self.network_window_left = IntVar(value=self.defineSettings[4])
        self.entry_2 = ttk.Entry(self.top, width=10, font="Times 14", textvariable=self.network_window_left)
        self.entry_2.place(x=500, y=200)
        self.label_5 = ttk.Label(self.top, text="Розмір правої частини вікна контексту: ", font="Times 14")
        self.label_5.place(x=30, y=250)
        self.network_window_right = IntVar(value=self.defineSettings[5])
```

```
        self.entry_3 = ttk.Entry(self.top, width=10, font="Times 14", textvariable=self.network_window_right)
        self.entry_3.place(x=500, y=250)
        self.label_6 = ttk.Label(self.top, text="Коефіцієнт оцінювання ідентичності до еталону: ", font="Times 14")
        self.label_6.place(x=800, y=50)
        self.etalon_literal_coeficient = StringVar(value=self.defineSettings[6])
        self.entry_4 = ttk.Entry(self.top, width=10, font="Times 14", textvariable=self.etalon_literal_coeficient)
        self.entry_4.place(x=1300, y=50)
        self.label_7 = ttk.Label(self.top, text="Коефіцієнт оцінювання ідентичності до корпусу: ", font="Times 14")
        self.label_7.place(x=800, y=100)
        self.corpus_literal_coeficient = StringVar(value=self.defineSettings[7])
        self.entry_5 = ttk.Entry(self.top, width=10, font="Times 14", textvariable=self.corpus_literal_coeficient)
        self.entry_5.place(x=1300, y=100)
        self.label_8 = ttk.Label(self.top, text="Коефіцієнт оцінювання синонімічності до еталону: ", font="Times 14")
        self.label_8.place(x=800, y=150)
        self.etalon_synonymus_coeficient = StringVar(value=self.defineSettings[8])
        self.entry_6 = ttk.Entry(self.top, width=10, font="Times 14", textvariable=self.etalon_synonymus_coeficient)
        self.entry_6.place(x=1300, y=150)
        self.label_9 = ttk.Label(self.top, text="Коефіцієнт оцінювання синонімічності до корпусу: ", font="Times 14")
        self.label_9.place(x=800, y=200)
        self.corpus_synonymus_coeficient = StringVar(value=self.defineSettings[9])
        self.entry_7 = ttk.Entry(self.top, width=10, font="Times 14", textvariable=self.corpus_synonymus_coeficient)
        self.entry_7.place(x=1300, y=200)
        self.label_10 = ttk.Label(self.top, text="Використання розміру текстів для оцінювання: ", font="Times 14")
        self.label_10.place(x=800, y=250)
        self.use_size_coeficient = IntVar(value=self.defineSettings[10])
        self.check_3 = Checkbutton(self.top, bootstyle=SUCCESS + "-round-toggle",
                                   variable=self.use_size_coeficient, onvalue=1, offvalue=0)
        self.check_3.place(x=1300, y=250)
        button_1 = Button(self.top, text="Підтвердити", bg='#00DDAA', font = "Times 18", width = 15, command =
self.submit)
        button_1.place(x=550, y=700)
        button_2 = Button(self.top, text="Назад", bg='#00DDAA', font = "Times 18", width = 15, command =
self.top.destroy)
        button_2.place(x=800, y=700)
        self.frame.pack()

    def choosePersonalCorpus(self):
        if self.is_personal_corpus.get() == 1:
            corpus_window = PersonalCorpusWindow(self.updatePersonalCorpus,
```

```
                                self.definePersonalCorpus())

    def updatePersonalCorpus(self,
                             personal_corpus):
        self.personal_corpus.setValues(personal_corpus)

    def definePersonalCorpus(self):
        return self.personal_corpus.getValues()

    def submit(self):
        self.updateSettings(self.n_gram_count.get(),
                            self.is_personal_corpus.get(),
                            self.definePersonalCorpus(),
                            self.is_integrated_corpus.get(),
                            self.network_window_left.get(),
                            self.network_window_right.get(),
                            self.etalon_literal_coeficient.get(),
                            self.corpus_literal_coeficient.get(),
                            self.etalon_synonymus_coeficient.get(),
                            self.corpus_synonymus_coeficient.get(),
                            self.use_size_coeficient.get())
        self.top.destroy()

class PersonalCorpusWindow:
    def __init__(self, updatePersonalCorpus, definePersonalCorpus):
        self.top = Toplevel()
        self.top.title("Вибір корпусу")
        self.top.focus_set()
        self.frame = Frame(self.top, width=700, height=780)
        self.updatePersonalCorpus = updatePersonalCorpus
        self.label_1 = Label(self.top, text = "Обрати шлях: ", font = "Times 14")
        self.label_1.place(x=30, y=50)
        self.path = StringVar(value=definePersonalCorpus[0])
        self.entry_1 = Entry(self.top, width=35, font="Times 14", textvariable=self.path)
        self.entry_1.place(x=30, y=100)
        self.textbox_1 = Text(self.top, height=20, width=58, font="Times 14")
        self.textbox_1.place(x=30, y=150)
        button_1 = Button(self.top, text="Пошук", bg='#00DDAA', font="Times 18", width=15, command=lambda:
self.chooseDirectory(self.entry_1, self.textbox_1))
```

```
        button_1.place(x=450, y=80)
        button_2 = Button(self.top, text="Підтвердити", bg='#00DDAA', font="Times 18", width=15,
command=self.submit)
        button_2.place(x=50, y=650)
        button_3 = Button(self.top, text="Назад", bg='#00DDAA', font="Times 18", width=15,
command=self.top.destroy)
        button_3.place(x=420, y=650)
        self.frame.pack()

    def chooseDirectory(self, entry, textbox):
        entry.delete(0, END)
        directoryname = askdirectory()
        entry.insert(0, directoryname)
        files = [f for f in listdir(directoryname) if isfile(join(directoryname, f))]
        textbox.delete("1.0", END)
        text_in_box = ""
        for i in range(len(files)):
            text_in_box += files[i] + "\n"
        textbox.insert('end', text_in_box)

    def submit(self):
        self.updatePersonalCorpus(self.path.get())
        self.top.destroy()

class PersonalCorpus:
    def __init__(self):
        self.path = StringVar()
        self.files = []

    def getValues(self):
        return self.path.get(), self.files

    def setValues(self, path):
        self.path.set(path)
        self.files = [f for f in listdir(self.path.get()) if isfile(join(self.path.get(), f))]

class InputWindow:
    def __init__(self, defineSettings):
        self.top = Toplevel()
```

```
        self.top.title("Вибір текстів")
        self.top.grab_set()
        self.frame = Frame(self.top, width = 1600, height = 780)
        self.defineSettings = defineSettings
        etalon = Content()
        self.label_1 = Label(self.top, text="Еталон", font = "Times 14")
        self.label_1.place(x=15, y=50)
        self.entry_1 = Entry(self.top, width = 30, font = "Times 18", textvariable = etalon.source)
        self.entry_1.place(x=15, y=100)
        self.textbox_1 = Text(self.top, height = 20, width = 65, font = "Times 14")
        self.textbox_1.place(x=15, y=200)
        button_1 = Button(self.top, text="Пошук", bg='#00DDAA', font = "Times 18", width = 10, command=lambda:
self.chooseFile(self.entry_1, self.textbox_1, etalon))
        button_1.place(x=450, y=90)
        comparable = Content()
        self.label_2 = Label(self.top, text="Текст для перевірки", font = "Times 14")
        self.label_2.place(x=815, y=50)
        self.entry_2 = Entry(self.top, width = 30, font = "Times 18", textvariable = comparable.source)
        self.entry_2.place(x=815, y=100)
        self.textbox_2 = Text(self.top, height = 20, width = 65, font = "Times 14")
        self.textbox_2.place(x=815, y=200)
        button_2 = Button(self.top, text="Пошук", bg='#00DDAA', font = "Times 18", width = 10, command=lambda:
self.chooseFile(self.entry_2, self.textbox_2, comparable))
        button_2.place(x=1250, y=90)
        button_3 = Button(self.top, text="Підтвердити", bg='#00DDAA', font = "Times 18", width = 15,
command=lambda: self.accept(etalon, comparable))
        button_3.place(x=550, y=700)
        button_4 = Button(self.top, text="Назад", bg='#00DDAA', font = "Times 18", width = 15, command =
self.top.destroy)
        button_4.place(x=800, y=700)
        self.frame.pack()

    def ReadFromPDF(self, filename, textbox, content):
        PDFFileObj = open(filename, "rb")
        PDFReader = PyPDF2.PdfFileReader(PDFFileObj)
        text = ""
        for pageNum in range(PDFReader.numPages):
            pageObj = PDFReader.getPage(pageNum)
            pageText = pageObj.extractText()
```

```
        text += "".join(pageText.split("\n"))
    content.text = text
    textbox.delete("1.0", END)
    textbox.insert('end', text)

def ReadFromDocx(self, filename, textbox, content):
    DocxFileObj = Document(filename)
    text = ""
    for paragraph in DocxFileObj.paragraphs:
        text += " " + paragraph.text
    content.text = text[1:]
    textbox.delete("1.0", END)
    textbox.insert('end', text[1:])

def chooseFile(self, entry, textbox, content):
    entry.delete(0, END)
    filename = askopenfilename()
    entry.insert(0, filename)
    if re.match(r'.*\.pdf$', filename):
        self.ReadFromPDF(filename, textbox, content)
    elif re.match(r'.*\.docx$', filename):
        self.ReadFromDocx(filename, textbox, content)
    return

def accept(self, etalon, comparable):
    personalCorpus = []
    if self.defineSettings[1] == 1:
        personalCorpus = self.getPersonalCorpusTexts()

    [literal_etalon_matrix, literal_comparable_matrix] = literal_etalon_copy.main(source="etalon",
                                                    etalon=etalon.text,
                                                    comparable=comparable.text,
                                                    n_gram_size=self.defineSettings[0])

    for i in range(len(personalCorpus)):
        literal_comparable_matrix = literal_personal_corpus_copy.main(source=personalCorpus[i].source,
                                                etalon=personalCorpus[i].text,
                                                comparable=comparable.text,
                                                comparable_matrix=literal_comparable_matrix,
```

```
                                        n_gram_size=self.defineSettings[0])

    [literal_etalon_matrix, literal_comparable_matrix] = etalon_synonymus.main(
        source="etalon",
        etalon=etalon.text,
        comparable=comparable.text,
        etalon_matrix=literal_etalon_matrix,
        comparable_matrix=literal_comparable_matrix,
        n_gram_size=self.defineSettings[0],
        left_wing=self.defineSettings[4],
        right_wing=self.defineSettings[5])

    for i in range(len(personalCorpus)):
        literal_comparable_matrix = personal_corpus_synonymus.main(source=personalCorpus[i].source,
                                        etalon=personalCorpus[i].text,
                                        comparable=comparable.text,
                                        comparable_matrix=literal_comparable_matrix,
                                        n_gram_size=self.defineSettings[0],
                                        left_wing=self.defineSettings[4],
                                        right_wing=self.defineSettings[5])

    mark = assessment.main(etalon_matrix=literal_etalon_matrix,
                comparable_matrix=literal_comparable_matrix,
                etalon_literal_coeficient=float(self.defineSettings[6]),
                corpus_literal_coeficient=float(self.defineSettings[7]),
                etalon_synonymus_coeficient=float(self.defineSettings[8]),
                corpus_synonymus_coeficient=float(self.defineSettings[9]),
                use_size_coeficient=self.defineSettings[10])
    result_window = ResultWindow()
    result_window.setText(etalon.text, comparable.text, mark)
    result_window.changeBgColor(comparable.text, literal_comparable_matrix)

def getPersonalCorpusTexts(self):
    textsCorpus = []
    for i in range(len(self.defineSettings[2][1])):
        filename = self.defineSettings[2][0] + "/" + self.defineSettings[2][1][i]
        textCorpus = Content()
        textCorpus.source = filename
        if re.match(r'.*\.pdf$', filename):
```

```
            textCorpus.text = self.ReadFromPDFCorpus(filename)
        elif re.match(r'.*\.docx$', filename):
            textCorpus.text = self.ReadFromDocxCorpus(filename)
        textsCorpus.append(textCorpus)
    return textsCorpus

def ReadFromPDFCorpus(self, filename):
    PDFFileObj = open(filename, "rb")
    PDFReader = PyPDF2.PdfFileReader(PDFFileObj)
    text = ""
    for pageNum in range(PDFReader.numPages):
        pageObj = PDFReader.getPage(pageNum)
        pageText = pageObj.extractText()
        text += "".join(pageText.split("\n"))
    return text

def ReadFromDocxCorpus(self, filename):
    DocxFileObj = Document(filename)
    text = ""
    for paragraph in DocxFileObj.paragraphs:
        text += " " + paragraph.text
    text = text[1:]
    return text

class Content:
def __init__(self):
    self.source = StringVar()
    self.text = StringVar()

class ResultWindow:
def __init__(self):
    top = Toplevel()
    top.title("Результат перевірки")
    top.grab_set()
    self.frame = Frame(top, width = 1600, height = 780)
    self.label_1 = Label(top, text="Еталон", font="Times 14")
    self.label_1.place(x=15, y=50)
    self.text_etalon = Text(top, width=65, height = 20, font = "Times 14")
    self.text_etalon.place(x=15, y=100)
```

```
        self.label_2 = Label(top, text="Текст для перевірки", font="Times 14")
        self.label_2.place(x=815, y=50)
        self.text_comparable = Text(top, width=65, height = 20, font = "Times 14")
        self.text_comparable.place(x=815, y=100)
        self.label_3 = Label(top, text="Рекомендований бал: ", font="Times 14")
        self.label_3.place(x=605, y=600)
        self.mark = StringVar()
        self.label_4 = Label(top, textvariable=self.mark, font="Times 14")
        self.label_4.place(x=850, y=600)
        button_1 = Button(top, text="Завершити", bg='#00DDAA', font = "Times 18", width = 15, command =
top.destroy)
        button_1.place(x=650, y=700)
        self.frame.pack()

    def setText(self, etalon, comparable, mark):
        self.text_etalon.insert(END, etalon)
        self.text_comparable.insert(END, comparable)
        self.mark.set(str(mark))

    def changeBgColor(self, comparable, comparable_matrix):
        for i in range(len(comparable_matrix)):
            if comparable_matrix[i][1] == 1:
                tag1 = "tag1" + str(i)
                self.text_comparable.tag_config(tag1, background="#00AA00")
                self.text_comparable.tag_add(tag1, "1." + str(comparable_matrix[i][5][0]), "1." +
str(comparable_matrix[i][5][1]))
                self.text_comparable.tag_raise(tag1)
                if i == 10:
                    hi = 0
                tag2 = "tag2" + str(i)
                self.text_comparable.tag_bind(tag1, "<Enter>", lambda event, tag=tag2, analog=comparable_matrix[i][6],
option=1: self.showInd(tag, analog, option))
                self.text_comparable.tag_bind(tag1, "<Leave>", lambda event, tag=tag2, analog=comparable_matrix[i][6],
option=0: self.showInd(tag, analog, option))
            if comparable_matrix[i][1] == 2:
                tag3 = "tag3" + str(i)
                self.text_comparable.tag_config(tag3, background="yellow")
                self.text_comparable.tag_add(tag3, "1." + str(comparable_matrix[i][5][0]), "1." +
str(comparable_matrix[i][5][1]))
```

```
            if comparable_matrix[i][1] == 3:
                tag4 = "tag4" + str(i)
                self.text_comparable.tag_config(tag4, background="cyan")
                self.text_comparable.tag_add(tag4, "1." + str(comparable_matrix[i][5][0]), "1." +
str(comparable_matrix[i][5][1]))
                self.text_comparable.tag_lower(tag4)
                self.text_comparable.tag_lower(tag4)

            if comparable_matrix[i][1] == 4:
                tag5 = "tag5" + str(i)
                self.text_comparable.tag_config(tag5, background="#FFA500")
                self.text_comparable.tag_add(tag5, "1." + str(comparable_matrix[i][5][0]),
                                             "1." + str(comparable_matrix[i][5][1]))
                self.text_comparable.tag_lower(tag5)
                self.text_comparable.tag_lower(tag5)
                self.text_comparable.tag_lower(tag5)

    def showInd(self, tag, analog, option):
        if option == 1:
            self.text_etalon.tag_config("start", background="yellow")
            self.text_etalon.tag_add("start", "1." + str(analog[0]), "1." + str(analog[1]))
        if option == 0:
            self.text_etalon.tag_remove("start", "1." + str(analog[0]), "1." + str(analog[1]))

def main():
    root = Tk()
    window = MainWindow(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Додаток Б

Код модулю буквального порівняння з еталоном

```
from nltk import word_tokenize
from nltk.util import ngrams
import re
from lemmagen3 import Lemmatizer
from collections import Counter

def sortPotential(list_to_sort):
    for i in range(len(list_to_sort)):
        if len(list_to_sort[i][2]) >= 2:
            completed_steps = 0
            while completed_steps < len(list_to_sort[i][2]):
                try_to_continue = True
                for j in range(len(list_to_sort[i][2])):
                    for k in range(j, len(list_to_sort[i][2])):
                        if list_to_sort[i][2][j][1] < list_to_sort[i][2][k][1]:
                            temp = list_to_sort[i][2][j]
                            list_to_sort[i][2][j] = list_to_sort[i][2][k]
                            list_to_sort[i][2][k] = temp
                            try_to_continue = False
                        else:
                            completed_steps += 1
                        if try_to_continue == False:
                            break
                    if try_to_continue == False:
                        break
                if try_to_continue == False:
                    completed_steps = 0
    return list_to_sort

def chooseBetter(etalon, comparable):
    for i in range(len(comparable)):
        if len(comparable[i][2]) >= 1:
            if len(etalon[comparable[i][2][0][0]][2]) >= 1:
                if etalon[comparable[i][2][0][0]][2][0][0] == i:
                    comparable[i][3] = comparable[i][2][0][0]
```

```
                etalon[comparable[i][2][0][0]][3] = i
                for j in range(len(comparable)):
                    for k in range(len(comparable[j][2])):
                        if comparable[j][2][k][0] == comparable[i][2][0][0] and i != j:
                            del comparable[j][2][k]
                            break
                for j in range(len(etalon)):
                    for k in range(len(etalon[j][2])):
                        if etalon[j][2][k][0] == etalon[comparable[i][2][0][0]][2][0][0] and comparable[i][2][0][0] != j:
                            del etalon[j][2][k]
                            break
                etalon[comparable[i][2][0][0]][2] = []
                comparable[i][2] = []
    return etalon, comparable

def forColor(comparable, n_grams_comparable):
    indexes_for_color = []
    current_position_global = 0
    for i in range(len(n_grams_comparable)):
        current_position_local = current_position_global
        start_index = None
        end_index = None
        for j in range(len(n_grams_comparable[i][0])):
            if j == 0:
                start_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                current_position_global = start_index + len(n_grams_comparable[i][0][j])
            elif j == (len(n_grams_comparable[i][0]) - 1):
                final_word_start_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                end_index = final_word_start_index + len(n_grams_comparable[i][0][j])
            else:
                middle_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                current_position_local = middle_index + len(n_grams_comparable[i][0][j])
        indexes_for_color.append([n_grams_comparable[i][0], start_index, end_index])
    return indexes_for_color

def comparison(source, etalon, comparable, number, replacement_dictionary, stop_words):

    lem_ukr = Lemmatizer("uk")
```

```
    tokens_comparable_for_color = word_tokenize(comparable)
    tokens_comparable_for_color = [token for token in tokens_comparable_for_color if re.match(r'\w+', token)]
    tokens_comparable_for_color = [token for token in tokens_comparable_for_color if
lem_ukr.lemmatize(token.lower()) not in stop_words]

    n_grams_comparable_for_color = ngrams(tokens_comparable_for_color, number)
    comparable_matrix_for_color = [[n_gram] for n_gram in n_grams_comparable_for_color]

    ind_for_color_comparable = forColor(comparable, comparable_matrix_for_color)

    ind_for_color_etalon = []
    if source == "etalon":
        tokens_etalon_for_color = word_tokenize(etalon)
        tokens_etalon_for_color = [token for token in tokens_etalon_for_color if re.match(r'\w+', token)]
        tokens_etalon_for_color = [token for token in tokens_etalon_for_color if token not in stop_words]

        n_grams_etalon_for_color = ngrams(tokens_etalon_for_color, number)
        etalon_matrix_for_color = [[n_gram] for n_gram in n_grams_etalon_for_color]

        ind_for_color_etalon = forColor(etalon, etalon_matrix_for_color)

    tokens_etalon = word_tokenize(etalon)
    tokens_etalon = [token for token in tokens_etalon if lem_ukr.lemmatize(token.lower()) not in stop_words]
    tokens_etalon = [lem_ukr.lemmatize(token.lower()) for token in tokens_etalon if re.match(r'\w+', token)]

    tokens_comparable = word_tokenize(comparable)
    tokens_comparable = [token for token in tokens_comparable if lem_ukr.lemmatize(token.lower()) not in stop_words]
    tokens_comparable = [lem_ukr.lemmatize(token.lower()) for token in tokens_comparable if re.match(r'\w+', token)]

    n_grams_etalon = ngrams(tokens_etalon, number)
    n_grams_comparable = ngrams(tokens_comparable, number)
    etalon_matrix = [[n_gram] for n_gram in n_grams_etalon]
    for row in etalon_matrix:
        row = row.extend((0, [], None,))
    comparable_matrix = [[n_gram] for n_gram in n_grams_comparable]
    for row in comparable_matrix:
        row = row.extend((0, [], None, [], [], []))

    #complicate
```

```
    for i in range(len(comparable_matrix)):
        for j in range(len(etalon_matrix)):
            isSimple = True
            for k in range(len(etalon_matrix[j][0])):
                if etalon_matrix[j][0][k] != comparable_matrix[i][0][k]:
                    if ((etalon_matrix[j][0][k] not in replacement_dictionary[0] and etalon_matrix[j][0][k] not in
replacement_dictionary[1]) or
                            (comparable_matrix[i][0][k] not in replacement_dictionary[0] and comparable_matrix[i][0][k] not in
replacement_dictionary[0]) or
                            (etalon_matrix[j][0][k] in replacement_dictionary[0] and comparable_matrix[i][0][k] in
replacement_dictionary[1] and replacement_dictionary[0].index(etalon_matrix[j][0][k]) !=
replacement_dictionary[1].index(comparable_matrix[i][0][k])) or
                            (comparable_matrix[i][0][k] in replacement_dictionary[0] and etalon_matrix[j][0][k] in
replacement_dictionary[1] and replacement_dictionary[0].index(comparable_matrix[i][0][k]) !=
replacement_dictionary[1].index(etalon_matrix[j][0][k]))):
                        isSimple = False
                difference_in_letters_etalon = len(etalon_matrix[j][0][k]) - sum((Counter(etalon_matrix[j][0][k]) &
Counter(comparable_matrix[i][0][k])).values())
                difference_in_letters_comparable = len(comparable_matrix[i][0][k]) - sum((Counter(etalon_matrix[j][0][k])
& Counter(comparable_matrix[i][0][k])).values())
                if (difference_in_letters_etalon <= 1 and difference_in_letters_comparable <= 1) and
len(etalon_matrix[j][0][k]) > 2 and len(comparable_matrix[i][0][k]) > 2:
                    if etalon_matrix[j][0][k][difference_in_letters_etalon:] ==
comparable_matrix[i][0][k][difference_in_letters_comparable:]:
                        if isSimple == False:
                            isSimple = True
            if isSimple:
                if source == "etalon":
                    comparable_matrix[i][1] = 1
                    etalon_matrix[j][1] = 1
                    comparable_matrix[i][2].append([j, 0])
                    etalon_matrix[j][2].append([i, 0])
                    comparable_matrix[i][4] = "etalon"
                elif source != "etalon":
                    comparable_matrix[i][1] = 2
                    etalon_matrix[j][1] = 2
                    comparable_matrix[i][2].append([j, 0])
                    etalon_matrix[j][2].append([i, 0])
                    comparable_matrix[i][4] = source
```

```
for i in range(len(comparable_matrix)):
    if len(comparable_matrix[i][2]) == 0:
        continue
    elif len(comparable_matrix[i][2]) == 1:
        continue
    elif len(comparable_matrix[i][2]) > 1:
        for j in range(len(comparable_matrix[i][2])):
            before = 0
            after = 0
            current = 1
            iter = comparable_matrix[i][2][j][0] - 1
            while iter != -1 and etalon_matrix[iter][1] == 1:
                before += 1
                iter -= 1
            iter = comparable_matrix[i][2][j][0] + 1
            while iter != len(etalon_matrix) and etalon_matrix[iter][1] == 1:
                after += 1
                iter += 1
            comparable_matrix[i][2][j][1] = before + current + after

            before = 0
            after = 0
            current = 0
            iter = i - 1
            while iter != -1 and comparable_matrix[iter][1] == 1:
                before += 1
                iter -= 1
            iter = i + 1
            while iter != len(comparable_matrix) and comparable_matrix[iter][1] == 1:
                after += 1
                iter += 1
            for k in range(len(etalon_matrix[comparable_matrix[i][2][j][0]][2])):
                if etalon_matrix[comparable_matrix[i][2][j][0]][2][k][0] == i:
                    etalon_matrix[comparable_matrix[i][2][j][0]][2][k][1] = before + current + after
                    break

etalon_matrix = sortPotential(etalon_matrix)
```

```
comparable_matrix = sortPotential(comparable_matrix)

isAllSingleEtalon = True
for i in range(len(etalon_matrix)):
    if len(etalon_matrix[i][2]) >= 1:
        isAllSingleEtalon = False
        break

isAllSingleComparable = True
for i in range(len(comparable_matrix)):
    if len(comparable_matrix[i][2]) >= 1:
        isAllSingleComparable = False
        break

hi = 0
while isAllSingleEtalon == False or isAllSingleComparable == False:
    etalon_matrix, comparable_matrix = chooseBetter(etalon_matrix, comparable_matrix)
    isAllSingleEtalon = True
    for i in range(len(etalon_matrix)):
        if len(etalon_matrix[i][2]) >= 1:
            isAllSingleEtalon = False
            break
    isAllSingleComparable = True
    for i in range(len(comparable_matrix)):
        if len(comparable_matrix[i][2]) >= 1:
            isAllSingleComparable = False
            break
    hi += 1
    if hi == 10:
        break

for i in range(len(comparable_matrix)):
    if comparable_matrix[i][1] == 1 and comparable_matrix[i][3] is None:
        comparable_matrix[i][1] = 0
        comparable_matrix[i][4] = []
    comparable_matrix[i][0] = comparable_matrix_for_color[i][0]
    comparable_matrix[i][5].extend((ind_for_color_comparable[i][1], ind_for_color_comparable[i][2]))
    if comparable_matrix[i][1] == 1:
```

```
            comparable_matrix[i][6].extend((ind_for_color_etalon[comparable_matrix[i][3]][1],
ind_for_color_etalon[comparable_matrix[i][3]][2]))

    return [etalon_matrix, comparable_matrix]

def readReplacementFile():
    replacement = [[], []]
    with open("replacements.txt", encoding='utf8') as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    for i in range(len(lines)):
        options = lines[i].split(" - ")
        replacement[0].append(options[0])
        replacement[1].append(options[1])
    return replacement

def readStopWordsFile():
    with open("stopwords_ua.txt", encoding="utf8") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    return lines

def main(source, etalon, comparable, n_gram_size):
    replacement_dictionary = readReplacementFile()
    stop_words = readStopWordsFile()
    return comparison(source, etalon, comparable, n_gram_size, replacement_dictionary, stop_words)

if __name__ == "__main__":
    main()
```

Додаток В

Код модулю буквального порівняння з корпусом

```
from nltk import word_tokenize
from nltk.util import ngrams
import re
from lemmagen3 import Lemmatizer
from collections import Counter

def sortPotential(list_to_sort):
    for i in range(len(list_to_sort)):
        if len(list_to_sort[i][2]) >= 2:
            completed_steps = 0
            while completed_steps < len(list_to_sort[i][2]):
                try_to_continue = True
                for j in range(len(list_to_sort[i][2])):
                    for k in range(j, len(list_to_sort[i][2])):
                        if list_to_sort[i][2][j][1] < list_to_sort[i][2][k][1]:
                            temp = list_to_sort[i][2][j]
                            list_to_sort[i][2][j] = list_to_sort[i][2][k]
                            list_to_sort[i][2][k] = temp
                            try_to_continue = False
                        else:
                            completed_steps += 1
                        if try_to_continue == False:
                            break
                    if try_to_continue == False:
                        break
                if try_to_continue == False:
                    completed_steps = 0
    return list_to_sort

def chooseBetter(source, etalon, comparable):
    for i in range(len(comparable)):
        if len(comparable[i][2]) >= 1:
            if len(etalon[comparable[i][2][0][0]][2]) >= 1:
                if etalon[comparable[i][2][0][0]][2][0][0] == i and comparable[i][2][0][2] == source:
```

```
            if comparable[i][4] != "etalon" and len(comparable[i][4]) != 0 and comparable[i][4][1] <
comparable[i][2][0][1]:
                comparable[i][3] = comparable[i][2][0][0]
                comparable[i][4] = comparable[i][2][0]
                etalon[comparable[i][2][0][0]][3] = i
            elif comparable[i][4] != "etalon" and len(comparable[i][4]) == 0:
                comparable[i][3] = comparable[i][2][0][0]
                comparable[i][4] = comparable[i][2][0]
                etalon[comparable[i][2][0][0]][3] = i
            for j in range(len(comparable)):
                for k in range(len(comparable[j][2])):
                    if comparable[j][2][k][0] == comparable[i][4][0] and i != j and comparable[j][2][k][2] ==
comparable[i][4][2]:
                        del comparable[j][2][k]
                        break
            for j in range(len(etalon)):
                for k in range(len(etalon[j][2])):
                    if etalon[j][2][k][0] == etalon[comparable[i][2][0][0]][2][0][0] and comparable[i][2][0][0] != j:
                        del etalon[j][2][k]
                        break
            etalon[comparable[i][2][0][0]][2] = []
            comparable[i][2] = []
    return etalon, comparable

def forColor(comparable, n_grams_comparable):
    indexes_for_color = []
    current_position_global = 0
    for i in range(len(n_grams_comparable)):
        current_position_local = current_position_global
        start_index = None
        end_index = None
        for j in range(len(n_grams_comparable[i][0])):
            if j == 0:
                start_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                current_position_global = start_index + len(n_grams_comparable[i][0][j])
            elif j == (len(n_grams_comparable[i][0]) - 1):
                final_word_start_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                end_index = final_word_start_index + len(n_grams_comparable[i][0][j])
            else:
```

```
            middle_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
            current_position_local = middle_index + len(n_grams_comparable[i][0][j])
        indexes_for_color.append([n_grams_comparable[i][0], start_index, end_index])
    return indexes_for_color

def comparison(source, etalon, comparable, comparable_matrix, number, replacement_dictionary, stop_words):

    lem_ukr = Lemmatizer("uk")

    tokens_comparable_for_color = word_tokenize(comparable)
    tokens_comparable_for_color = [token for token in tokens_comparable_for_color if re.match(r'\w+', token)]
    tokens_comparable_for_color = [token for token in tokens_comparable_for_color if
lem_ukr.lemmatize(token.lower()) not in stop_words]

    n_grams_comparable_for_color = ngrams(tokens_comparable_for_color, number)
    comparable_matrix_for_color = [[n_gram] for n_gram in n_grams_comparable_for_color]

    ind_for_color_comparable = forColor(comparable, comparable_matrix_for_color)

    tokens_etalon = word_tokenize(etalon)
    tokens_etalon = [lem_ukr.lemmatize(token.lower()) for token in tokens_etalon if re.match(r'\w+', token)]
    tokens_etalon = [token for token in tokens_etalon if token not in stop_words]
    tokens_comparable = word_tokenize(comparable)
    tokens_comparable = [lem_ukr.lemmatize(token.lower()) for token in tokens_comparable if re.match(r'\w+', token)]
    tokens_comparable = [token for token in tokens_comparable if token not in stop_words]

    n_grams_etalon = ngrams(tokens_etalon, number)
    n_grams_comparable = ngrams(tokens_comparable, number)
    etalon_matrix = [[n_gram] for n_gram in n_grams_etalon]
    for row in etalon_matrix:
        row = row.extend((0, [], None))
    temp_matrix = [[n_gram] for n_gram in n_grams_comparable]
    for i in range(len(comparable_matrix)):
        comparable_matrix[i][0] = temp_matrix[i][0]

    #complicate
    for i in range(len(comparable_matrix)):
        for j in range(len(etalon_matrix)):
            isSimple = True
```

```
        for k in range(len(etalon_matrix[j][0])):
            if etalon_matrix[j][0][k] != comparable_matrix[i][0][k]:
                if ((etalon_matrix[j][0][k] not in replacement_dictionary[0] and etalon_matrix[j][0][k] not in
replacement_dictionary[1]) or
                    (comparable_matrix[i][0][k] not in replacement_dictionary[0] and comparable_matrix[i][0][k] not in
replacement_dictionary[0]) or
                    (etalon_matrix[j][0][k] in replacement_dictionary[0] and comparable_matrix[i][0][k] in
replacement_dictionary[1] and replacement_dictionary[0].index(etalon_matrix[j][0][k]) !=
replacement_dictionary[1].index(comparable_matrix[i][0][k])) or
                    (comparable_matrix[i][0][k] in replacement_dictionary[0] and etalon_matrix[j][0][k] in
replacement_dictionary[1] and replacement_dictionary[0].index(comparable_matrix[i][0][k]) !=
replacement_dictionary[1].index(etalon_matrix[j][0][k]))):
                        isSimple = False
                difference_in_letters_etalon = len(etalon_matrix[j][0][k]) - sum((Counter(etalon_matrix[j][0][k]) &
Counter(comparable_matrix[i][0][k])).values())
                difference_in_letters_comparable = len(comparable_matrix[i][0][k]) - sum((Counter(etalon_matrix[j][0][k])
& Counter(comparable_matrix[i][0][k])).values())
                if (difference_in_letters_etalon <= 1 and difference_in_letters_comparable <= 1) and
len(etalon_matrix[j][0][k]) > 2 and len(comparable_matrix[i][0][k]) > 2:
                    if etalon_matrix[j][0][k][difference_in_letters_etalon:] ==
comparable_matrix[i][0][k][difference_in_letters_comparable:]:
                        if isSimple == False:
                            isSimple = True
                if isSimple == False:
                    break
        if isSimple:
            if source == "etalon":
                comparable_matrix[i][1] = 1
                etalon_matrix[j][1] = 1
                comparable_matrix[i][2].append([j, 0])
                etalon_matrix[j][2].append([i, 0])
                comparable_matrix[i][4] = "etalon"
            elif source != "etalon":
                if comparable_matrix[i][1] != 1:
                    comparable_matrix[i][1] = 2
                    etalon_matrix[j][1] = 2
                    comparable_matrix[i][2].append([j, 0, source])
                    etalon_matrix[j][2].append([i, 0, source])
```

```
for i in range(len(comparable_matrix)):
    if comparable_matrix[i][1] == 2:
        if len(comparable_matrix[i][2]) == 0:
            continue
        elif len(comparable_matrix[i][2]) == 1:
            continue
        elif len(comparable_matrix[i][2]) > 1:
            for j in range(len(comparable_matrix[i][2])):
                before = 0
                after = 0
                current = 1
                iter = comparable_matrix[i][2][j][0] - 1
                while iter != -1 and etalon_matrix[iter][1] == 2:
                    before += 1
                    iter -= 1
                iter = comparable_matrix[i][2][j][0] + 1
                while iter != len(etalon_matrix) and etalon_matrix[iter][1] == 2:
                    after += 1
                    iter += 1
                comparable_matrix[i][2][j][1] = before + current + after

                before = 0
                after = 0
                current = 0
                iter = i - 1
                while iter != -1 and comparable_matrix[iter][1] == 2:
                    before += 1
                    iter -= 1
                iter = i + 1
                while iter != len(comparable_matrix) and comparable_matrix[iter][1] == 2:
                    after += 1
                    iter += 1
                for k in range(len(etalon_matrix[comparable_matrix[i][2][j][0]][2])):
                    if etalon_matrix[comparable_matrix[i][2][j][0]][2][k][0] == i:
                        etalon_matrix[comparable_matrix[i][2][j][0]][2][k][1] = before + current + after
                        break

isAllSingleEtalon = True
for i in range(len(etalon_matrix)):
```

```
    if len(etalon_matrix[i][2]) >= 1:
        isAllSingleEtalon = False
        break

isAllSingleComparable = True
for i in range(len(comparable_matrix)):
    if len(comparable_matrix[i][2]) >= 1:
        isAllSingleComparable = False
        break

hi = 0
while isAllSingleEtalon == False or isAllSingleComparable == False:
    etalon_matrix, comparable_matrix = chooseBetter(source, etalon_matrix, comparable_matrix)
    isAllSingleEtalon = True
    for i in range(len(etalon_matrix)):
        if len(etalon_matrix[i][2]) >= 1:
            isAllSingleEtalon = False
            break
    isAllSingleComparable = True
    for i in range(len(comparable_matrix)):
        if len(comparable_matrix[i][2]) >= 1:
            isAllSingleComparable = False
            break
    hi += 1
    if hi == 10:
        break

for i in range(len(comparable_matrix)):
    if comparable_matrix[i][1] == 2 and len(comparable_matrix[i][4]) == 0:
        comparable_matrix[i][1] = 0
    comparable_matrix[i][0] = comparable_matrix_for_color[i][0]
    if comparable_matrix[i][1] == 2:
        comparable_matrix[i][5].extend((ind_for_color_comparable[i][1], ind_for_color_comparable[i][2]))

positives = 0
for row in comparable_matrix:
    positives += row[1]

correct_percent = round((positives / len(comparable_matrix))*100, 1)
```

```
    return comparable_matrix

def readReplacementFile():
    replacement = [[], []]
    with open("replacements.txt", encoding='utf8') as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    for i in range(len(lines)):
        options = lines[i].split(" - ")
        replacement[0].append(options[0])
        replacement[1].append(options[1])
    return replacement

def readStopWordsFile():
    with open("stopwords_ua.txt", encoding="utf8") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    return lines

def main(source, etalon, comparable, comparable_matrix, n_gram_size):
    replacement_dictionary = readReplacementFile()
    stop_words = readStopWordsFile()
    return comparison(source, etalon, comparable, comparable_matrix, n_gram_size, replacement_dictionary,
stop_words)

if __name__ == "__main__":
    main()
```

Додаток Г

Код модулю синонімічного порівняння з еталоном

```
from nltk import word_tokenize
from nltk.util import ngrams
import re
from lemmagen3 import Lemmatizer
from collections import Counter
import network_initialization
import network_prediction_simple

def sortPotential(list_to_sort):
    for i in range(len(list_to_sort)):
        if len(list_to_sort[i][2]) >= 2:
            completed_steps = 0
            while completed_steps < len(list_to_sort[i][2]):
                try_to_continue = True
                for j in range(len(list_to_sort[i][2])):
                    for k in range(j, len(list_to_sort[i][2])):
                        if list_to_sort[i][2][j][1] < list_to_sort[i][2][k][1]:
                            temp = list_to_sort[i][2][j]
                            list_to_sort[i][2][j] = list_to_sort[i][2][k]
                            list_to_sort[i][2][k] = temp
                            try_to_continue = False
                        else:
                            completed_steps += 1
                        if try_to_continue == False:
                            break
                    if try_to_continue == False:
                        break
                if try_to_continue == False:
                    completed_steps = 0
    return list_to_sort


def chooseBetter(etalon, comparable):
    for i in range(len(comparable)):
        if len(comparable[i][2]) >= 1:
```

```
            if len(etalon[comparable[i][2][0][0]][2]) >= 1:
                if etalon[comparable[i][2][0][0]][2][0][0] == i:
                    comparable[i][3] = comparable[i][2][0][0]
                    etalon[comparable[i][2][0][0]][3] = i
                    for j in range(len(comparable)):
                        for k in range(len(comparable[j][2])):
                            if comparable[j][2][k][0] == comparable[i][2][0][0] and i != j:
                                del comparable[j][2][k]
                                break
                    for j in range(len(etalon)):
                        for k in range(len(etalon[j][2])):
                            if etalon[j][2][k][0] == etalon[comparable[i][2][0][0]][2][0][0] and comparable[i][2][0][
                                0] != j:
                                del etalon[j][2][k]
                                break
                    etalon[comparable[i][2][0][0]][2] = []
                    comparable[i][2] = []
    return etalon, comparable


def forColor(comparable, n_grams_comparable):
    indexes_for_color = []
    current_position_global = 0
    for i in range(len(n_grams_comparable)):
        current_position_local = current_position_global
        start_index = None
        end_index = None
        for j in range(len(n_grams_comparable[i][0])):
            if j == 0:
                start_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                current_position_global = start_index + len(n_grams_comparable[i][0][j])
            elif j == (len(n_grams_comparable[i][0]) - 1):
                final_word_start_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                end_index = final_word_start_index + len(n_grams_comparable[i][0][j])
            else:
                middle_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                current_position_local = middle_index + len(n_grams_comparable[i][0][j])
        indexes_for_color.append([n_grams_comparable[i][0], start_index, end_index])
    return indexes_for_color
```

```
def comparison(source, etalon, comparable, etalon_matrix, comparable_matrix, number, replacement_dictionary,
stop_words, left_wing, right_wing,
            vocabluary, weights):
    lem_ukr = Lemmatizer("uk")

    tokens_comparable_for_color = word_tokenize(comparable)
    tokens_comparable_for_color = [token for token in tokens_comparable_for_color if re.match(r'\w+', token)]
    tokens_comparable_for_color = [token for token in tokens_comparable_for_color if
                                   lem_ukr.lemmatize(token.lower()) not in stop_words]

    n_grams_comparable_for_color = ngrams(tokens_comparable_for_color, number)
    comparable_matrix_for_color = [[n_gram] for n_gram in n_grams_comparable_for_color]

    ind_for_color_comparable = forColor(comparable, comparable_matrix_for_color)

    ind_for_color_etalon = []
    if source == "etalon":
        tokens_etalon_for_color = word_tokenize(etalon)
        tokens_etalon_for_color = [token for token in tokens_etalon_for_color if re.match(r'\w+', token)]
        tokens_etalon_for_color = [token for token in tokens_etalon_for_color if token not in stop_words]

        n_grams_etalon_for_color = ngrams(tokens_etalon_for_color, number)
        etalon_matrix_for_color = [[n_gram] for n_gram in n_grams_etalon_for_color]

        ind_for_color_etalon = forColor(etalon, etalon_matrix_for_color)

    tokens_etalon = word_tokenize(etalon)
    tokens_etalon = [token for token in tokens_etalon if lem_ukr.lemmatize(token.lower()) not in stop_words]
    tokens_etalon = [lem_ukr.lemmatize(token.lower()) for token in tokens_etalon if re.match(r'\w+', token)]

    tokens_comparable = word_tokenize(comparable)
    tokens_comparable = [token for token in tokens_comparable if lem_ukr.lemmatize(token.lower()) not in stop_words]
    tokens_comparable = [lem_ukr.lemmatize(token.lower()) for token in tokens_comparable if re.match(r'\w+', token)]

    n_grams_etalon = ngrams(tokens_etalon, number)
    n_grams_comparable = ngrams(tokens_comparable, number)
    temp_etalon_matrix = [[n_gram] for n_gram in n_grams_etalon]
```

```
for i in range(len(etalon_matrix)):
    etalon_matrix[i][0] = temp_etalon_matrix[i][0]
temp_comparable_matrix = [[n_gram] for n_gram in n_grams_comparable]
for i in range(len(comparable_matrix)):
    comparable_matrix[i][0] = temp_comparable_matrix[i][0]

# complicate
for i in range(len(comparable_matrix)):
    if comparable_matrix[i][1] != 1 and comparable_matrix[i][1] != 2:
        for j in range(len(etalon_matrix)):
            if etalon_matrix[j][1] != 1:
                isSimple = True
                for k in range(len(etalon_matrix[j][0])):
                    if etalon_matrix[j][0][k] != comparable_matrix[i][0][k]:
                        if ((etalon_matrix[j][0][k] not in replacement_dictionary[0] and etalon_matrix[j][0][k] not in
                             replacement_dictionary[1]) or
                                (comparable_matrix[i][0][k] not in replacement_dictionary[0] and comparable_matrix[i][0][
                                    k] not in replacement_dictionary[0]) or
                                (etalon_matrix[j][0][k] in replacement_dictionary[0] and comparable_matrix[i][0][k] in
                                 replacement_dictionary[1] and replacement_dictionary[0].index(etalon_matrix[j][0][k]) !=
                                 replacement_dictionary[1].index(comparable_matrix[i][0][k])) or
                                (comparable_matrix[i][0][k] in replacement_dictionary[0] and etalon_matrix[j][0][k] in
                                 replacement_dictionary[1] and replacement_dictionary[0].index(
                                         comparable_matrix[i][0][k]) != replacement_dictionary[1].index(
                                         etalon_matrix[j][0][k]))):
                            isSimple = False
                        difference_in_letters_etalon = len(etalon_matrix[j][0][k]) - sum(
                            (Counter(etalon_matrix[j][0][k]) & Counter(comparable_matrix[i][0][k])).values())
                        difference_in_letters_comparable = len(comparable_matrix[i][0][k]) - sum(
                            (Counter(etalon_matrix[j][0][k]) & Counter(comparable_matrix[i][0][k])).values())
                        if (difference_in_letters_etalon <= 1 and difference_in_letters_comparable <= 1) and len(
                                etalon_matrix[j][0][k]) > 2 and len(comparable_matrix[i][0][k]) > 2:
                            if etalon_matrix[j][0][k][difference_in_letters_etalon:] == comparable_matrix[i][0][k][
                                                                        difference_in_letters_comparable:]:
                                if isSimple == False:
                                    isSimple = True
                        if isSimple == False:
                            context = []
                            beta = j
```

```
                    fita = k
                    step_left = 0
                    while step_left < left_wing:
                        if fita > 0:
                            fita -= 1
                            context.append(etalon_matrix[beta][0][fita])
                        elif fita == 0 and beta > 0:
                            beta -= 0
                            context.append(etalon_matrix[beta][0][fita])
                        step_left += 1
                    beta = j
                    fita = k
                    step_right = 0
                    while step_right < right_wing:
                        if fita < number - 1:
                            fita += 1
                            context.append(etalon_matrix[beta][0][fita])
                        elif fita == number - 1 and beta < len(etalon_matrix) - 1:
                            beta += 1
                            context.append(etalon_matrix[beta][0][fita])
                        step_right += 1
                    synonymus_result = network_prediction_simple.prediction(context, comparable_matrix[i][0][k],
vocabluary, weights)
                        isSimple = synonymus_result
                    if isSimple == False:
                        break
                if isSimple:
                    if source == "etalon":
                        comparable_matrix[i][1] = 3
                        etalon_matrix[j][1] = 3
                        comparable_matrix[i][2].append([j, 0])
                        etalon_matrix[j][2].append([i, 0])
                        comparable_matrix[i][4] = "etalon"

    for i in range(len(comparable_matrix)):
        if len(comparable_matrix[i][2]) == 0:
            continue
        elif len(comparable_matrix[i][2]) == 1:
            continue
```

```
        elif len(comparable_matrix[i][2]) > 1:
            for j in range(len(comparable_matrix[i][2])):
                before = 0
                after = 0
                current = 1
                iter = comparable_matrix[i][2][j][0] - 1
                while iter != -1 and (etalon_matrix[iter][1] == 1 or etalon_matrix[iter][1] == 3):
                    before += 1
                    iter -= 1
                iter = comparable_matrix[i][2][j][0] + 1
                while iter != len(etalon_matrix) and (etalon_matrix[iter][1] == 1 or etalon_matrix[iter][1] == 3):
                    after += 1
                    iter += 1
                comparable_matrix[i][2][j][1] = before + current + after

                before = 0
                after = 0
                current = 0
                iter = i - 1
                while iter != -1 and (comparable_matrix[iter][1] == 1 or comparable_matrix[iter][1] == 3):
                    before += 1
                    iter -= 1
                iter = i + 1
                while iter != len(comparable_matrix) and (comparable_matrix[iter][1] == 1 or comparable_matrix[iter][1] ==
3):
                    after += 1
                    iter += 1
                for k in range(len(etalon_matrix[comparable_matrix[i][2][j][0]][2])):
                    if etalon_matrix[comparable_matrix[i][2][j][0]][2][k][0] == i:
                        etalon_matrix[comparable_matrix[i][2][j][0]][2][k][1] = before + current + after
                        break

    etalon_matrix = sortPotential(etalon_matrix)

    comparable_matrix = sortPotential(comparable_matrix)

    isAllSingleEtalon = True
    for i in range(len(etalon_matrix)):
        if len(etalon_matrix[i][2]) >= 1:
```

```
        isAllSingleEtalon = False
        break

isAllSingleComparable = True
for i in range(len(comparable_matrix)):
    if len(comparable_matrix[i][2]) >= 1:
        isAllSingleComparable = False
        break

hi = 0
while isAllSingleEtalon == False or isAllSingleComparable == False:
    etalon_matrix, comparable_matrix = chooseBetter(etalon_matrix, comparable_matrix)
    isAllSingleEtalon = True
    for i in range(len(etalon_matrix)):
        if len(etalon_matrix[i][2]) >= 1:
            isAllSingleEtalon = False
            break
    isAllSingleComparable = True
    for i in range(len(comparable_matrix)):
        if len(comparable_matrix[i][2]) >= 1:
            isAllSingleComparable = False
            break
    hi += 1
    if hi == 10:
        break

for i in range(len(comparable_matrix)):
    if comparable_matrix[i][1] == 3 and comparable_matrix[i][3] is None:
        comparable_matrix[i][1] = 0
        comparable_matrix[i][4] = []
    comparable_matrix[i][0] = comparable_matrix_for_color[i][0]
    if comparable_matrix[i][1] == 3:
        comparable_matrix[i][5].extend((ind_for_color_comparable[i][1], ind_for_color_comparable[i][2]))
    if comparable_matrix[i][1] == 3:
        comparable_matrix[i][6].extend(
            (ind_for_color_etalon[comparable_matrix[i][3]][1], ind_for_color_etalon[comparable_matrix[i][3]][2]))

return [etalon_matrix, comparable_matrix]
```

```
def readReplacementFile():
    replacement = [[], []]
    with open("replacements.txt", encoding='utf8') as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    for i in range(len(lines)):
        options = lines[i].split(" - ")
        replacement[0].append(options[0])
        replacement[1].append(options[1])
    return replacement

def readStopWordsFile():
    with open("stopwords_ua.txt", encoding="utf8") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    return lines

def main(source, etalon, comparable, etalon_matrix, comparable_matrix, n_gram_size, left_wing, right_wing):
    replacement_dictionary = readReplacementFile()
    stop_words = readStopWordsFile()
    [vocabluary, weights] = network_initialization.network_initialize()
    return comparison(source, etalon, comparable, etalon_matrix, comparable_matrix, n_gram_size,
replacement_dictionary, stop_words, left_wing, right_wing, vocabluary, weights)

if __name__ == "__main__":
    main()
```

Додаток Ґ

Код модулю синонімічного порівняння з корпусом

```
from nltk import word_tokenize
from nltk.util import ngrams
import re
from lemmagen3 import Lemmatizer
from collections import Counter
import network_prediction_simple
import network_initialization

def sortPotential(list_to_sort):
    for i in range(len(list_to_sort)):
        if len(list_to_sort[i][2]) >= 2:
            completed_steps = 0
            while completed_steps < len(list_to_sort[i][2]):
                try_to_continue = True
                for j in range(len(list_to_sort[i][2])):
                    for k in range(j, len(list_to_sort[i][2])):
                        if list_to_sort[i][2][j][1] < list_to_sort[i][2][k][1]:
                            temp = list_to_sort[i][2][j]
                            list_to_sort[i][2][j] = list_to_sort[i][2][k]
                            list_to_sort[i][2][k] = temp
                            try_to_continue = False
                        else:
                            completed_steps += 1
                        if try_to_continue == False:
                            break
                    if try_to_continue == False:
                        break
                if try_to_continue == False:
                    completed_steps = 0
    return list_to_sort

def chooseBetter(source, etalon, comparable):
    for i in range(len(comparable)):
        if len(comparable[i][2]) >= 1:
            if len(etalon[comparable[i][2][0][0]][2]) >= 1:
```

```
            if etalon[comparable[i][2][0][0]][2][0][0] == i and comparable[i][2][0][2] == source:
                if comparable[i][4] != "etalon" and len(comparable[i][4]) != 0 and comparable[i][4][1] <
comparable[i][2][0][1]:
                    comparable[i][3] = comparable[i][2][0][0]
                    comparable[i][4] = comparable[i][2][0]
                    etalon[comparable[i][2][0][0]][3] = i
                elif comparable[i][4] != "etalon" and len(comparable[i][4]) == 0:
                    comparable[i][3] = comparable[i][2][0][0]
                    comparable[i][4] = comparable[i][2][0]
                    etalon[comparable[i][2][0][0]][3] = i
                for j in range(len(comparable)):
                    for k in range(len(comparable[j][2])):
                        if comparable[j][2][k][0] == comparable[i][4][0] and i != j and comparable[j][2][k][2] ==
comparable[i][4][2]:
                            del comparable[j][2][k]
                            break
                for j in range(len(etalon)):
                    for k in range(len(etalon[j][2])):
                        if etalon[j][2][k][0] == etalon[comparable[i][2][0][0]][2][0][0] and comparable[i][2][0][0] != j:
                            del etalon[j][2][k]
                            break
                etalon[comparable[i][2][0][0]][2] = []
                comparable[i][2] = []
    return etalon, comparable

def forColor(comparable, n_grams_comparable):
    indexes_for_color = []
    current_position_global = 0
    for i in range(len(n_grams_comparable)):
        current_position_local = current_position_global
        start_index = None
        end_index = None
        for j in range(len(n_grams_comparable[i][0])):
            if j == 0:
                start_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                current_position_global = start_index + len(n_grams_comparable[i][0][j])
            elif j == (len(n_grams_comparable[i][0]) - 1):
                final_word_start_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                end_index = final_word_start_index + len(n_grams_comparable[i][0][j])
```

```
            else:
                middle_index = comparable.find(n_grams_comparable[i][0][j], current_position_local)
                current_position_local = middle_index + len(n_grams_comparable[i][0][j])
        indexes_for_color.append([n_grams_comparable[i][0], start_index, end_index])
    return indexes_for_color

def comparison(source, etalon, comparable, comparable_matrix, number, replacement_dictionary, stop_words,
left_wing, right_wing, vocabluary, weights):

    lem_ukr = Lemmatizer("uk")

    tokens_comparable_for_color = word_tokenize(comparable)
    tokens_comparable_for_color = [token for token in tokens_comparable_for_color if re.match(r'\w+', token)]
    tokens_comparable_for_color = [token for token in tokens_comparable_for_color if
lem_ukr.lemmatize(token.lower()) not in stop_words]

    n_grams_comparable_for_color = ngrams(tokens_comparable_for_color, number)
    comparable_matrix_for_color = [[n_gram] for n_gram in n_grams_comparable_for_color]

    ind_for_color_comparable = forColor(comparable, comparable_matrix_for_color)

    tokens_etalon = word_tokenize(etalon)
    tokens_etalon = [lem_ukr.lemmatize(token.lower()) for token in tokens_etalon if re.match(r'\w+', token)]
    tokens_etalon = [token for token in tokens_etalon if token not in stop_words]
    tokens_comparable = word_tokenize(comparable)
    tokens_comparable = [lem_ukr.lemmatize(token.lower()) for token in tokens_comparable if re.match(r'\w+', token)]
    tokens_comparable = [token for token in tokens_comparable if token not in stop_words]

    n_grams_etalon = ngrams(tokens_etalon, number)
    n_grams_comparable = ngrams(tokens_comparable, number)
    etalon_matrix = [[n_gram] for n_gram in n_grams_etalon]
    for row in etalon_matrix:
        row = row.extend((0, [], None))
    temp_matrix = [[n_gram] for n_gram in n_grams_comparable]
    for i in range(len(comparable_matrix)):
        comparable_matrix[i][0] = temp_matrix[i][0]

    #complicate
    for i in range(len(comparable_matrix)):
```

```
    if comparable_matrix[i][1] != 1 and comparable_matrix[i][1] != 2 and comparable_matrix[i][1] != 3:
        for j in range(len(etalon_matrix)):
            isSimple = True
            for k in range(len(etalon_matrix[j][0])):
                if etalon_matrix[j][0][k] != comparable_matrix[i][0][k]:
                    if ((etalon_matrix[j][0][k] not in replacement_dictionary[0] and etalon_matrix[j][0][k] not in
replacement_dictionary[1]) or
                            (comparable_matrix[i][0][k] not in replacement_dictionary[0] and comparable_matrix[i][0][k] not in
replacement_dictionary[0]) or
                            (etalon_matrix[j][0][k] in replacement_dictionary[0] and comparable_matrix[i][0][k] in
replacement_dictionary[1] and replacement_dictionary[0].index(etalon_matrix[j][0][k]) !=
replacement_dictionary[1].index(comparable_matrix[i][0][k])) or
                            (comparable_matrix[i][0][k] in replacement_dictionary[0] and etalon_matrix[j][0][k] in
replacement_dictionary[1] and replacement_dictionary[0].index(comparable_matrix[i][0][k]) !=
replacement_dictionary[1].index(etalon_matrix[j][0][k]))):
                        isSimple = False
                    difference_in_letters_etalon = len(etalon_matrix[j][0][k]) - sum((Counter(etalon_matrix[j][0][k]) &
Counter(comparable_matrix[i][0][k])).values())
                    difference_in_letters_comparable = len(comparable_matrix[i][0][k]) -
sum((Counter(etalon_matrix[j][0][k]) & Counter(comparable_matrix[i][0][k])).values())
                    if (difference_in_letters_etalon <= 1 and difference_in_letters_comparable <= 1) and
len(etalon_matrix[j][0][k]) > 2 and len(comparable_matrix[i][0][k]) > 2:
                        if etalon_matrix[j][0][k][difference_in_letters_etalon:] ==
comparable_matrix[i][0][k][difference_in_letters_comparable:]:
                            if isSimple == False:
                                isSimple = True
                    if isSimple == False:
                        context = []
                        beta = j
                        fita = k
                        step_left = 0
                        while step_left < left_wing:
                            if fita > 0:
                                fita -= 1
                                context.append(etalon_matrix[beta][0][fita])
                            elif fita == 0 and beta > 0:
                                beta -= 0
                                context.append(etalon_matrix[beta][0][fita])
                            step_left += 1
```

```
                    beta = j
                    fita = k
                    step_right = 0
                    while step_right < right_wing:
                        if fita < number - 1:
                            fita += 1
                            context.append(etalon_matrix[beta][0][fita])
                        elif fita == number - 1 and beta < len(etalon_matrix) - 1:
                            beta += 1
                            context.append(etalon_matrix[beta][0][fita])
                        step_right += 1
                    synonymus_result = network_prediction_simple.prediction(context, comparable_matrix[i][0][k],
                                                                             vocabluary, weights)
                    isSimple = synonymus_result
                if isSimple == False:
                    break
                if isSimple == False:
                    break
            if isSimple:
                if source == "etalon":
                    comparable_matrix[i][1] = 3
                    etalon_matrix[j][1] = 3
                    comparable_matrix[i][2].append([j, 0])
                    etalon_matrix[j][2].append([i, 0])
                    comparable_matrix[i][4] = "etalon"
                elif source != "etalon":
                    if comparable_matrix[i][1] != 1 and comparable_matrix[i][1] != 2 and comparable_matrix[i][1] != 3:
                        comparable_matrix[i][1] = 4
                        etalon_matrix[j][1] = 4
                        comparable_matrix[i][2].append([j, 0, source])
                        etalon_matrix[j][2].append([i, 0, source])

for i in range(len(comparable_matrix)):
    if comparable_matrix[i][1] == 2:
        if len(comparable_matrix[i][2]) == 0:
            continue
        elif len(comparable_matrix[i][2]) == 1:
            continue
        elif len(comparable_matrix[i][2]) > 1:
```

```
            for j in range(len(comparable_matrix[i][2])):
                before = 0
                after = 0
                current = 1
                iter = comparable_matrix[i][2][j][0] - 1
                while iter != -1 and (etalon_matrix[iter][1] == 2 or etalon_matrix[iter][1] == 4):
                    before += 1
                    iter -= 1
                iter = comparable_matrix[i][2][j][0] + 1
                while iter != len(etalon_matrix) and (etalon_matrix[iter][1] == 2 or etalon_matrix[iter][1] == 4):
                    after += 1
                    iter += 1
                comparable_matrix[i][2][j][1] = before + current + after

                before = 0
                after = 0
                current = 0
                iter = i - 1
                while iter != -1 and (comparable_matrix[iter][1] == 2 or comparable_matrix[iter][1] == 4):
                    before += 1
                    iter -= 1
                iter = i + 1
                while iter != len(comparable_matrix) and (comparable_matrix[iter][1] == 2 or comparable_matrix[iter][1]
== 4):
                    after += 1
                    iter += 1
                for k in range(len(etalon_matrix[comparable_matrix[i][2][j][0]][2])):
                    if etalon_matrix[comparable_matrix[i][2][j][0]][2][k][0] == i:
                        etalon_matrix[comparable_matrix[i][2][j][0]][2][k][1] = before + current + after
                        break

    isAllSingleEtalon = True
    for i in range(len(etalon_matrix)):
        if len(etalon_matrix[i][2]) >= 1:
            isAllSingleEtalon = False
            break

    isAllSingleComparable = True
    for i in range(len(comparable_matrix)):
```

```
        if len(comparable_matrix[i][2]) >= 1:
            isAllSingleComparable = False
            break

    hi = 0
    while isAllSingleEtalon == False or isAllSingleComparable == False:
        etalon_matrix, comparable_matrix = chooseBetter(source, etalon_matrix, comparable_matrix)
        isAllSingleEtalon = True
        for i in range(len(etalon_matrix)):
            if len(etalon_matrix[i][2]) >= 1:
                isAllSingleEtalon = False
                break
        isAllSingleComparable = True
        for i in range(len(comparable_matrix)):
            if len(comparable_matrix[i][2]) >= 1:
                isAllSingleComparable = False
                break
        hi += 1
        if hi == 10:
            break

    for i in range(len(comparable_matrix)):
        if comparable_matrix[i][1] == 4 and len(comparable_matrix[i][4]) == 0:
            comparable_matrix[i][1] = 0
        comparable_matrix[i][0] = comparable_matrix_for_color[i][0]
        if comparable_matrix[i][1] == 4:
            comparable_matrix[i][5].extend((ind_for_color_comparable[i][1], ind_for_color_comparable[i][2]))

    return comparable_matrix

def readReplacementFile():
    replacement = [[], []]
    with open("replacements.txt", encoding='utf8') as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    for i in range(len(lines)):
        options = lines[i].split(" - ")
        replacement[0].append(options[0])
```

```
        replacement[1].append(options[1])
    return replacement

def readStopWordsFile():
    with open("stopwords_ua.txt", encoding="utf8") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    return lines

def main(source, etalon, comparable, comparable_matrix, n_gram_size, left_wing, right_wing):
    replacement_dictionary = readReplacementFile()
    stop_words = readStopWordsFile()
    [vocabluary, weights] = network_initialization.network_initialize()
    return comparison(source, etalon, comparable, comparable_matrix, n_gram_size, replacement_dictionary,
stop_words, left_wing, right_wing, vocabluary, weights)

if __name__ == "__main__":
    main()
```

Додаток Д

Код модулю ініціалізації нейронної мережі

```
import os.path
import re
import PyPDF2
from docx import Document
import spacy

def ReadFromPDF(filename):
    PDFFileObj = open(filename, "rb")
    PDFReader = PyPDF2.PdfFileReader(PDFFileObj)
    text = ""
    for pageNum in range(PDFReader.numPages):
        pageObj = PDFReader.getPage(pageNum)
        pageText = pageObj.extractText()
        text += "".join(pageText.split("\n"))
    return text

def ReadFromDocx(filename):
    DocxFileObj = Document(filename)
    text = ""
    for paragraph in DocxFileObj.paragraphs:
        text += " " + paragraph.text
    return text[1:]

def NER_enteties(text):
    NER_ukr_lg = spacy.load("uk_core_news_lg")
    NER_text = NER_ukr_lg(text)
    NER_enteties_list = []
    for word in NER_text.ents:
        NER_enteties_list.append(word.text)
    return NER_enteties_list

def network_initialize():
    NER_enteties_list = []
    if os.path.isfile("./NER_enteties.txt"):
        with open("NER_enteties.txt") as f:
```

```
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    ind = 0
    while ind < len(lines):
        if lines[ind] == "":
            del lines[ind]
        else:
            ind += 1
    for i in range(len(lines)):
        NER_enteties_list.append(lines[i])
else:
    print("Помилка: Файл NER_enteties.txt не існує")

vocabluary = []
if os.path.isfile("./vocabluary.txt"):
    with open("vocabluary.txt") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    ind = 0
    while ind < len(lines):
        if lines[ind] == "":
            del lines[ind]
        else:
            ind += 1
    for i in range(len(lines)):
        vocabluary.append(lines[i])
else:
    print("Помилка: Файл vocabluary.txt не існує")

weights = []
if os.path.isfile("./weights.txt"):
    with open("weights.txt") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    ind = 0
    while ind < len(lines):
```

```
        if lines[ind] == "":
            del lines[ind]
        else:
            ind += 1
    layer = []
    for i in range(len(lines)):
        splited = re.findall("\S+", lines[i])
        for j in range(len(splited)):
            if splited[j] == "{" or splited[j] == "}":
                continue
            else:
                layer.append(float(splited[j]))
        weights.append([layer[j] for j in range(len(layer))])
        layer.clear()

return [vocabluary, weights]
```

Додаток Е

Код модулю тренування нейронної мережі

```
import os.path
import re
import PyPDF2
from docx import Document
from lemmagen3 import Lemmatizer
from nltk import word_tokenize
import spacy
import math

def sigmoid_activation(x):
    return 1/(1 + math.e**(-x))

def sigmoid_derivative(x):
    return (1/(1 + math.e**(-x))) * (1 - 1/(1 + math.e**(-x)))

def Sort(arr):
    return sorted(arr, reverse=True, key=lambda x: x[1])

def ReadFromPDF(filename):
    PDFFileObj = open(filename, "rb")
    PDFReader = PyPDF2.PdfFileReader(PDFFileObj)
    text = ""
    for pageNum in range(PDFReader.numPages):
        pageObj = PDFReader.getPage(pageNum)
        pageText = pageObj.extractText()
        text += "".join(pageText.split("\n"))
    return text

def ReadFromDocx(filename):
    DocxFileObj = Document(filename)
    text = ""
    for paragraph in DocxFileObj.paragraphs:
        text += " " + paragraph.text
    return text[1:]
```

```
def lstm_training(input_output, vocabluary, weights):

    step = 1000000
    total_final_input = []
    total_final_output = []
    total_inputs = []
    total_desired_outputs = []
    total_soft_max = []
    total_cross_entropy = []

    #forward propagation
    for i in range(len(input_output)):
        input_values = []
        output_value = None
        for j in range(len(input_output[i])):
            if len(input_output[i][j]) == 2 and input_output[i][j][1] == "main_object":
                output_value = input_output[i][j][0]
            else:
                input_values.append(input_output[i][j])

        desired_output_result = None
        input_vocabluary_values = []
        for j in range(len(vocabluary)):
            input_vocabluary_value = 0
            if vocabluary[j] in input_values:
                input_vocabluary_value = 1

            input_vocabluary_values.append(input_vocabluary_value)
            if vocabluary[j] == output_value:
                desired_output_result = j

        total_inputs.append(input_vocabluary_values)
        total_desired_outputs.append(desired_output_result)

        final_layer_results = []
        final_layer_input = []
        for j in range(len(vocabluary)):
            input = 0
            for k in range(len(vocabluary)):
```

```
            input += weights[k][j] * input_vocabluary_values[k]
        final_layer_input.append(input)
        cell_result = sigmoid_activation(input)
        final_layer_results.append(cell_result)

    total_final_input.append(final_layer_input)
    total_final_output.append(final_layer_results)

#soft_max
for i in range(len(total_final_output)):
    soft_max_experiment = []
    total_denumerator = sum([math.e**total_final_output[i][j] for j in range(len(total_final_output[i]))])
    for j in range(len(total_final_output[i])):
        soft_max_experiment.append(math.e**total_final_output[i][j] / total_denumerator)
    total_soft_max.append(soft_max_experiment)

#cross-entropy
for i in range(len(total_soft_max)):
    cross_entropy_experiment = []
    for j in range(len(total_soft_max[i])):
        if j == total_desired_outputs[i]:
            total_cross_entropy.append(-(math.log(total_soft_max[i][j])))
            break

mean_cross_entropy_error = sum(total_cross_entropy) / len(total_cross_entropy)

dCEdOUT = []
for i in range(len(total_inputs)):
    dCEdOUT_row = []
    for j in range(len(total_soft_max[i])):
        if j == total_desired_outputs[i]:
            dCEdOUT_row.append(total_soft_max[i][j] - 1)
        else:
            dCEdOUT_row.append(total_soft_max[i][j])
    dCEdOUT.append(dCEdOUT_row)

full_delta_weights = [[0.0 for k in range(len(vocabluary))] for j in range(len(vocabluary))]
new_weights = [[weights[j][k] for k in range(len(vocabluary))] for j in range(len(vocabluary))]
```

```
    #backpropagation
    for i in range(len(total_inputs)):
        for j in range(len(total_final_output[i])):
            for k in range(len(total_inputs[i])):
                delta_weight = dCEdOUT[i][j] * total_final_output[i][j] * (1 - total_final_output[i][j]) * total_inputs[i][k]
                full_delta_weights[k][j] += delta_weight
    for i in range(len(full_delta_weights)):
        for j in range(len(full_delta_weights[i])):
            full_delta_weights[i][j] /= len(total_inputs)
            new_weights[i][j] -= full_delta_weights[i][j] * step

    return [mean_cross_entropy_error, new_weights]

def lstm_prediction(input, vocabluary, weights, biases):
    input_neurons = [0 for i in range(len(vocabluary))]
    for i in range(len(vocabluary)):
        if vocabluary[i] in input:
            input_neurons[i] = 1

    results = []
    for j in range(len(vocabluary)):
        input = 0
        for k in range(len(vocabluary)):
            input += weights[k][j] * input_neurons[j]
        input += biases[j]
        cell_result = sigmoid_activation(input)
        results.append([vocabluary[j], cell_result])

    final_result = Sort(results)
    top_5 = []
    worst_5 = []
    for i in range(len(final_result)):
        if final_result[i][1] > 0.9:
            top_5.append(final_result[i])
        if final_result[i][1] < 0.1:
            worst_5.append(final_result[i])

    return [top_5, worst_5]
```

```
def NER_enteties(text):
    NER_ukr_lg = spacy.load("uk_core_news_lg")
    NER_text = NER_ukr_lg(text)
    NER_enteties_list = []
    for word in NER_text.ents:
        NER_enteties_list.append(word.text)
    return NER_enteties_list

def optimize(window_size_left, window_size_right, stopwords):

    lem_ukr = Lemmatizer("uk")
    NER_enteties_list = []
    if os.path.isfile("./NER_enteties.txt"):
        with open("NER_enteties.txt") as f:
            lines = f.readlines()
        for i in range(len(lines)):
            lines[i] = lines[i][:-1]
        ind = 0
        while ind < len(lines):
            if lines[ind] == "":
                del lines[ind]
            else:
                ind += 1
        for i in range(len(lines)):
            NER_enteties_list.append(lines[i])
        file_ent = open("NER_enteties.txt", "a")
    else:
        file_ent = open("NER_enteties.txt", "w+")

    used_for_training = []
    if os.path.isfile("./used-for-training.txt"):
        with open("used-for-training.txt") as f:
            lines = f.readlines()
        for i in range(len(lines)):
            lines[i] = lines[i][:-1]
        ind = 0
        while ind < len(lines):
            if lines[ind] == "":
                del lines[ind]
```

```
        else:
            ind += 1
    for i in range(len(lines)):
        used_for_training.append(lines[i])
    file_train = open("used-for-training.txt", "a")
else:
    file_train = open("used-for-training.txt", "w+")

vocabluary = []
if os.path.isfile("./vocabluary.txt"):
    with open("vocabluary.txt") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    ind = 0
    while ind < len(lines):
        if lines[ind] == "":
            del lines[ind]
        else:
            ind += 1
    for i in range(len(lines)):
        vocabluary.append(lines[i])
    file_vocabluary = open("vocabluary.txt", "a")
else:
    file_vocabluary = open("vocabluary.txt", "w+")

training_data = []

if True:
    standart_directory = os.listdir("./biology5")
    for i in range(len(standart_directory)):
        if re.match(r'.*\.pdf$', "./biology5/" + standart_directory[i]):
            text = ReadFromPDF("./biology5/" + standart_directory[i])
        elif re.match(r'.*\.docx$', "./biology5/" + standart_directory[i]):
            text = ReadFromDocx("./biology5/" + standart_directory[i])

        token_with_ne = ""
        tokens = word_tokenize(text)
        NER_ents = NER_enteties(text=text)
```

```
for j in range(len(NER_ents)):
    if lem_ukr.lemmatize(NER_ents[j]) not in NER_enteties_list:
        NER_enteties_list.append(lem_ukr.lemmatize(NER_ents[j]))
        file_ent.write(lem_ukr.lemmatize(NER_ents[j]) + "\n")
for j in range(len(tokens)):
    if re.match(r'\w+', tokens[j]):
        if lem_ukr.lemmatize(tokens[j]) in NER_enteties_list:
            continue
        tokens_appendable = lem_ukr.lemmatize(tokens[j].lower())
        if tokens_appendable == "не":
            token_with_ne = tokens_appendable
            continue
        if token_with_ne == "не":
            tokens_appendable = token_with_ne + " " + tokens_appendable
            token_with_ne = ""
        if tokens_appendable not in vocabluary and tokens_appendable not in stopwords:
            vocabluary.append(tokens_appendable)
            file_vocabluary.write(tokens_appendable + "\n")

tokens = [token for token in tokens if re.match(r'\w+', token)]
tokens = [token for token in tokens if token.lower() not in stopwords]
tokens = [lem_ukr.lemmatize(token) for token in tokens]
tokens = [token for token in tokens if token not in NER_enteties_list]
token_with_ne = ""
it_tok = 0
while it_tok < len(tokens):
    if token_with_ne == "не":
        tokens[it_tok-1] = token_with_ne + " " + tokens[it_tok]
        del tokens[it_tok]
        token_with_ne = ""
        it_tok -= 1
    if tokens[it_tok].lower() == "не":
        token_with_ne = tokens[it_tok].lower()
    it_tok += 1
tokens = [token.lower() for token in tokens]
tokens = [token for token in tokens if token not in stopwords]

for j in range(len(tokens)):
    input_object = []
```

```
            for k in range(1, window_size_left + 1):
                if j - k >= 0:
                    input_object.append(tokens[j-k])
                else:
                    break
            input_object.append([tokens[j], "main_object"])
            for k in range(1, window_size_right + 1):
                if j + k < len(tokens):
                    input_object.append(tokens[j+k])
                else:
                    break
            training_data.append(input_object)

        used_for_training.append(os.path.abspath(standart_directory[i]))
        file_train.write(os.path.abspath(standart_directory[i]) + "\n")

file_vocabluary.close()

weights = []
if os.path.isfile("./weights.txt"):
    with open("weights.txt") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    ind = 0
    while ind < len(lines):
        if lines[ind] == "":
            del lines[ind]
        else:
            ind += 1
    layer = []
    this_layer = False
    for i in range(len(lines)):
        splited = re.findall("\S+", lines[i])
        layer_element_weights = []
        for j in range(len(splited)):
            if splited[j] == "{" or splited[j] == "}":
                continue
            else:
```

```
            layer.append(float(splited[j]))
        weights.append([layer[j] for j in range(len(layer))])
        layer.clear()
else:
    file = open("weights.txt", "w+")
    number_of_layers = 1
    number_of_inputs = len(vocabluary)
    total_text = ""
    for i in range(number_of_layers):
        text_to_append = "{ "
        for j in range(number_of_inputs):
            input_string = ""
            for k in range(len(vocabluary)):
                element_input = 0.1
                input_string += "{:.5}".format(element_input) + " "
            if j != number_of_inputs - 1:
                input_string += "\n"
            text_to_append += input_string
        text_to_append += "}\n"
        total_text += text_to_append + "\n"
    file.write(total_text)
    file.close()

epochs = 10
for i in range(epochs):
    [error, weights] = lstm_training(input_output=training_data,
                                     vocabluary=vocabluary,
                                     weights=weights)

    print("EPOCH: " + str(i+1) + " ERROR: " + str(error))

file = open("weights.txt", "w")
number_of_inputs = len(vocabluary)
total_text = "{ "
for j in range(number_of_inputs):
    input_string = ""
    for k in range(len(vocabluary)):
        element_input = weights[j][k]
        input_string += "{:.5}".format(element_input) + " "
```

```
            if j != number_of_inputs - 1:
                input_string += "\n"
            total_text += input_string
        total_text += "}\n"
        file.write(total_text)
        file.close()

def readStopWordsFile():
    with open("stopwords_ua.txt", encoding="utf8") as f:
        lines = f.readlines()
    for i in range(len(lines)):
        lines[i] = lines[i][:-1]
    return lines

def main():
    stop_words = readStopWordsFile()
    window_size_left = 1
    window_size_right = 1
    optimize(window_size_left, window_size_right, stop_words)

if __name__ == "__main__":
    main()
```

Додаток Є

Код модулю нейронної мережі для прогнозування

```
import math

def sigmoid_activation(x):
    return 1/(1 + math.e**(-x))

def Sort(arr):
    return sorted(arr, reverse=True, key=lambda x: x[1])

def prediction(input, target, vocabluary, weights):
    input_neurons = [0 for i in range(len(vocabluary))]
    for i in range(len(vocabluary)):
        if vocabluary[i] in input:
            input_neurons[i] = 1

    results = []
    for j in range(len(vocabluary)):
        input_activ = 0
        for k in range(len(vocabluary)):
            input_activ += weights[k][j] * input_neurons[k]
        cell_result = sigmoid_activation(input_activ)
        results.append([vocabluary[j], cell_result])

    soft_max_experiment = []
    total_denumerator = sum([math.e ** results[i][1] for i in range(len(results))])
    for i in range(len(results)):
        soft_max_experiment.append(math.e ** results[i][1] / total_denumerator)
    soft_results = [[results[i][0], soft_max_experiment[i]] for i in range(len(soft_max_experiment))]

    final_result = Sort(soft_results)
    synonymus = [final_result[i][0] for i in range(7) if final_result[i][0] not in input]

    is_synonymus = False
    if target in synonymus:
        is_synonymus = True
```

```
return is_synonymus
```

Додаток Ж

Код модулю алгоритму оцінювання

```
def main(etalon_matrix, comparable_matrix, etalon_literal_coeficient, corpus_literal_coeficient, etalon_synonymus_coeficient, corpus_synonymus_coeficient, use_size_coeficient):
    etalon_variant = len(etalon_matrix)
    comparable_variant = len(comparable_matrix)
    if use_size_coeficient:
        if etalon_variant >= comparable_variant:
            dependance = comparable_variant / etalon_variant
        else:
            dependance = etalon_variant / comparable_variant
    else:
        dependance = 1
    total = 0
    for i in range(len(comparable_matrix)):
        if comparable_matrix[i][1] == 1:
            total += 1 * etalon_literal_coeficient
        elif comparable_matrix[i][1] == 2:
            total += 1 * corpus_literal_coeficient
        elif comparable_matrix[i][1] == 3:
            total += 1 * etalon_synonymus_coeficient
        elif comparable_matrix[i][1] == 4:
            total += 1 * corpus_synonymus_coeficient
        else:
            total += 0
    total /= len(comparable_matrix)
    mark = round(100 * total * dependance)
    return mark

if __name__ == "__main__":
    main()
```