Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

# ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
**за першим (бакалаврським) рівнем вищої освіти**
на тему
**“Програмне забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж”**



Виконала здобувачка вищої освіти
IV курсу, групи КІ-22МБ
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Варченко І.В.
«____» _____ 2025 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є. В.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

**Міністерство освіти і науки України**
**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *бакалавр*
Галузь знань *12 "Інформаційні технології"*
Спеціальність *123 Комп'ютерна інженерія*
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри**
**д.т.н., проф.**
**_____О.А.Смірнов**
« __ » _____ 2025 року

# З А В Д А Н Н Я
## НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

*Варченко Інні Володимирівні*
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, професор*
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу *№48-02 від 17.01.2025 року*

2. Строк подання студентом роботи до захисту *. .2025 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

*1. Призначення та область використання.*
*2.Перегляд аналогічних існуючих систем.*
*3. Опис і обґрунтування проектних рішень.*
*4. Етапи програмування системи.*
*5. Впровадження системи в промислову експлуатацію.*
*6. Висновки*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

| | |
|---|---|
| *Структурна схема системи* | *1 аркуш* |
| *Функціональна схема системи* | *1 аркуш* |
| *Діаграма процесів* | *1 аркуш* |
| *Блок-схема алгоритму роботи додатку* | *2 аркуша* |

6. Дата видачі завдання « » 2025 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної бакалаврської роботи | Строк виконання етапів кваліфікаційної бакалаврської роботи | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем | 10.03.2025 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 15.03.2025 р. | |
| 3. | Розробка моделі компонента | 20.03.2025 р. | |
| 4. | Розробка структур даних | 25.03.2025 р. | |
| 5. | Розробка алгоритмів зв’язку та відображення | 30.03.2025 р. | |
| 6. | Програмування алгоритмів | 10.04.2025 р. | |
| 7. | Оформлення ПЗ | 17.04.2025 р. | |
| 8. | Попередній захист роботи | 24.05.2025 р. | |
| | | | |
| | | | |
| | | | |
| | | | |

**Студент** _____ **Варченко І.В**
( підпис ) (прізвище та ініціали)

**Керівник роботи** _____ **Мелешко Є.В.**
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

**Варченко І.В. Програмне забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.**

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж.

Метою роботи є створення нейронної мережі для аналізу текстів на наявність інформаційно-психологічних впливів.

Результат роботи – програмна реалізація нейронної мережі для аналізу текстів на наявність інформаційно-психологічних впливів.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на рушії Unity з використанням мов програмування C# та Python.

**Ключові слова**: комп'ютерна інженерія, штучний інтелект, аналіз текстів, інформаційно-психологічний вплив

# ABSTRACT

**Varchenko I.V. Software for analyzing text for informational and psychological influences using neural networks. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.**

This qualification bachelor's thesis has developed software for analyzing text for informational and psychological influences using neural networks.

The purpose of the work is to create an artificial intelligence system for analyzing texts for informational and psychological influences.

The result of the work is a software implementation of artificial intelligence for analyzing texts for informational and psychological influences.

In the process of working on the implementation of the system, a study of existing methods, algorithms and software tools was carried out. The software itself was developed and implemented, and all its components were described.

A convenient user interface was developed. Instructions for working with software tools are provided.

The program can be used on IBM PC architecture computers with Windows 10/11 OS.

The program was developed on the Unity engine using the C# and Python programming languages.

**Keywords:** computer engineering, artificial intelligence, text analysis, informational and psychological impact

ЗМІСТ





| Вим. | Арк. | № докум. | Підп. | Дата | ВКРБ-123.25.0064.00.00.ПЗ | | |
|---|---|---|---|---|---|---|---|
| Розроб. | | Варченко І.В.. | | | Програмне забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж | Літ. | Аркуш 1 / Аркушів 53 |
| Перев. | | Мелешко Є.В. | | | | Б | |
| Н.контр. | | Коваленко А.С. | | | | ЦНТУ КІ-22мб | |
| Затв. | | Смірнов О.А. | | | | | |

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ШІ – Штучний інтелект.

ПЗ – Програмне забезпечення.

API - Application programming interface.

NLP - Natural Language Processing.

ML - Machine Learning.

TTR - Type/token ratio.

ReLU - Rectified Linear Unit.

VADER - Valence Aware Dictionary and Sentiment Reasoner.

Нейромережа – це комп'ютерна система, що імітує роботу людського мозку для розв'язання складних задач, таких як розпізнавання образів, обробка мови або передбачення результатів.

Сентимент-аналіз – це процес використання комп’ютерного програмного забезпечення для з’ясування думок або почуттів людей щодо чогось із написаного.

Лексикон – це спеціальні словники або списки слів для аналізу настроїв

# ВСТУП

Актуальність теми дипломної роботи полягає в необхідності розробки технологій, здатних автоматично оцінювати тексти на наявність інформаційно-психологічних впливів. В умовах зростання інформаційних загроз, поширення дезінформації та спрямованого впливу на суспільну свідомість важливо мати інструменти, що дозволяють аналізувати тексти на глибшому рівні.

Традиційні методи аналізу текстів не завжди здатні розпізнати приховані смислові конструкції, стилістичні особливості або лексичні прийоми, які можуть бути використані для формування певного враження у читачів. Крім того, значні обсяги текстової інформації, що поширюються через медіа та соціальні платформи, ускладнюють швидку та об’єктивну оцінку змісту.

Тому розробка програмного забезпечення, яке забезпечить автоматизований аналіз текстових даних, є важливим напрямком досліджень. Така система має надавати можливість виявлення прихованих смислів, оцінки структури тексту та його потенційного впливу, що сприятиме підвищенню якості аналізу інформації та покращенню розуміння її змісту.

**Мета й завдання дослідження.** Метою роботи є програмне забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих методів аналізу текстів на наявність інформаційно-психологічних впливів

– Розробка нейронної мережі для аналізу текстів на наявність інформаційно-психологічних впливів.

– Програмна реалізація нейронної мережі для аналізу текстів на наявність інформаційно-психологічних впливів.

**Практична цінність отриманих результатів** що розроблене програмне забезпечення дозволяє автоматизувати процес аналізу текстів на наявність інформаційно-психологічних впливів. Це сприяє ефективному виявленню потенційних загроз, що можуть впливати на громадську думку та поведінку користувачів.

Отже, розробка та впровадження системи штучного інтелекту для аналізу текстів є актуальним завданням, яке потребує постійного вдосконалення та адаптації до сучасних інформаційних викликів і вирішувалося у цій кваліфікаційній роботі.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

## 1.1 Призначення системи

Програмне забезпечення призначене для оцінки текстів на предмет інформаційно-психологічного впливу. Воно використовує методи аналізу тональності, стилістичних особливостей та частотності ключових слів для виявлення елементів пропаганди, маніпуляцій та прихованого психологічного тиску.

Система працює на основі алгоритмів обробки природної мови, що дозволяє аналізувати тексти різної тематики та стилістики. Вона виявляє використання маніпулятивних технік, оцінює рівень емоційного забарвлення тексту та визначає ступінь його впливу на читача. Зокрема, аналізуються такі характеристики, як риторичні прийоми, узагальнення, перебільшення, емоційні тригери та підміна понять.

Основна мета такого аналізу – допомогти ідентифікувати тексти, що можуть містити пропаганду, маніпуляції чи інші форми психологічного впливу. Результати оцінки можуть бути використані для фільтрації контенту, маркування підозрілих матеріалів або подальшого детального аналізу експертами.

Важливою перевагою такого підходу є можливість значного скорочення часу, необхідного для перевірки інформації, без втрати якості аналізу. Автоматизація процесу дозволяє швидко обробляти великі масиви текстів, зменшуючи потребу в ручному аналізі та мінімізуючи людський фактор при оцінюванні контенту. Це особливо важливо в умовах інформаційного перенасичення, коли швидкість реагування на потенційно шкідливі матеріали відіграє критичну роль.

## 1.2 Область застосування

Програмне забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів може застосовуватися у широкому спектрі галузей, де важливо швидко і точно оцінювати текстовий контент.

Однією з ключових сфер використання є кібербезпека. Аналіз текстової інформації дозволяє своєчасно виявляти можливі інформаційні загрози, такі як маніпулятивні або дезінформаційні матеріали, що можуть бути використані у гібридних атаках. Використання такого програмного забезпечення сприяє захисту інформаційного простору від шкідливого впливу та допомагає зменшити ризики, пов'язані з поширенням недостовірної інформації.

Ще однією важливою сферою застосування є журналістика та медіа-аналітика. Інструменти аналізу допомагають журналістам, редакторам і дослідникам розпізнавати інформаційні маніпуляції, оцінювати об'єктивність новин і виявляти тенденційні або пропагандистські матеріали.

Також система може бути використана в державних структурах, які займаються моніторингом інформаційного простору та протидією дезінформації. Аналіз інформаційного впливу дозволяє оцінювати громадську реакцію на певні події, що дає змогу своєчасно реагувати на потенційні загрози та приймати обґрунтовані рішення.

У сфері корпоративної безпеки подібні технології допомагають компаніям захищати свою репутацію, аналізуючи публічні відгуки, коментарі та інформаційні атаки конкурентів. Автоматизований аналіз текстів дозволяє швидко реагувати на негативні кампанії, виявляти спроби дестабілізації та оцінювати інформаційні ризики для бізнесу.

Загалом, застосування такої системи дозволяє автоматизувати аналіз текстової інформації, мінімізувати вплив людського фактору та підвищити ефективність реагування на інформаційні загрози.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

## 2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Семантичний аналіз тексту Istio оцінює надлишок ключових слів, воду та спам. Пошукові системи визначають якість і релевантність тексту за словами та фразами, які він містить.

Якщо текст містить відповідну кількість релевантних ключових фраз, пошукові системи оцінять його позитивно. Водянисті статті та статті з недостатньою кількістю ключових слів не відображатимуться на першій сторінці результатів пошуку. Тексти, переповнені ключовими словами, розглядаються як спам, і пошукові системи рідко їх показують. Сервіс показує:

- Щільність ключових слів і їх співвідношення до ядра тексту,
- Кількість слів і символів, з пробілами і без них,
- Лексика, за загальною кількістю статей і основних одиниць
- Щільність слів із зазначенням топ-10 найуживаніших
- Мова тексту та орієнтовна тематика
- Вода

Семантичний аналіз тексту Istio автоматично підраховує кількість символів і оцінює надлишок і воду. Сервіс виділяє ключові слова та воду та малює зручну діаграму частот.

Індекс надмірного спаму показує, наскільки текст перенасичений ключовими словами. Чим більше ключових слів, тим вищий індекс. Пошукові системи будуть оцінювати такі тексти як неякісні і не виводити їх на першу сторінку.

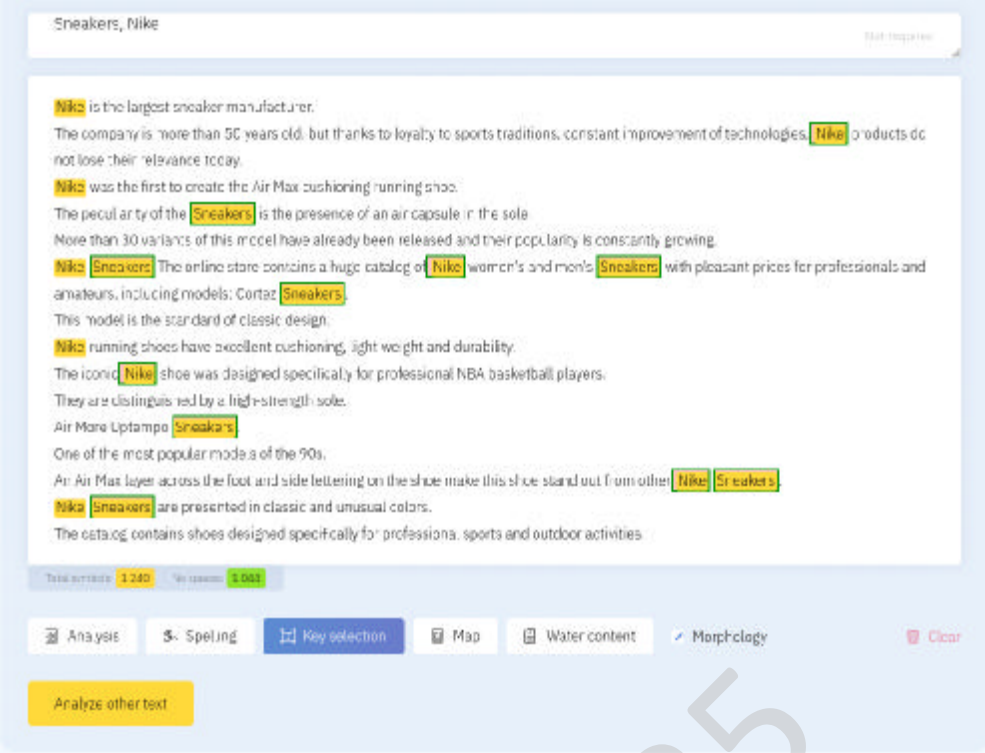


Рисунок 2.1 – Пошук ключових слів

Кількість ключових слів і їх співвідношення до загального тексту і до його ядра буде показано внизу сторінки.

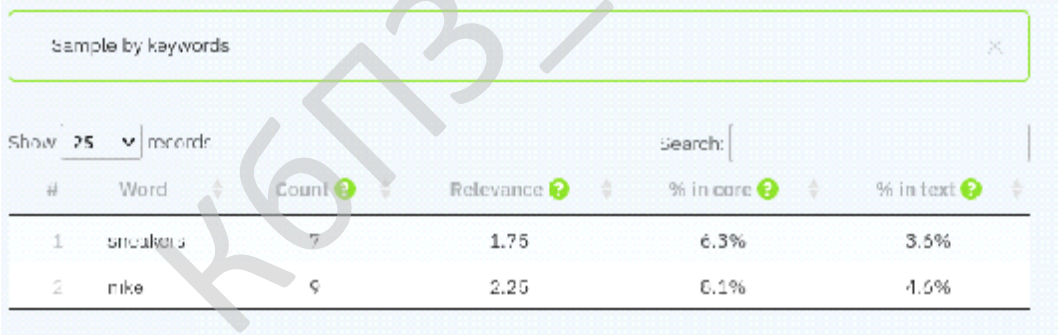


Рисунок 2.2 – Розрахунок співвідношень

Perspective API - використовує моделі машинного навчання для виявлення образливих коментарів. Моделі оцінюють фразу на основі відчутного впливу, який текст може мати на розмову. Розробники та видавці можуть використовувати цю оцінку, щоб надати відгук коментаторам, допомогти модераторам легше переглядати коментарі або допомогти читачам відфільтрувати мову.

Perspective API безкоштовний і доступний для використання арабською, китайською, чеською, голландською, англійською, французькою, німецькою, гінді, хінгліш, індонезійською, італійською, японською, корейською, польською, португальською, російською, іспанською та шведською мовами.

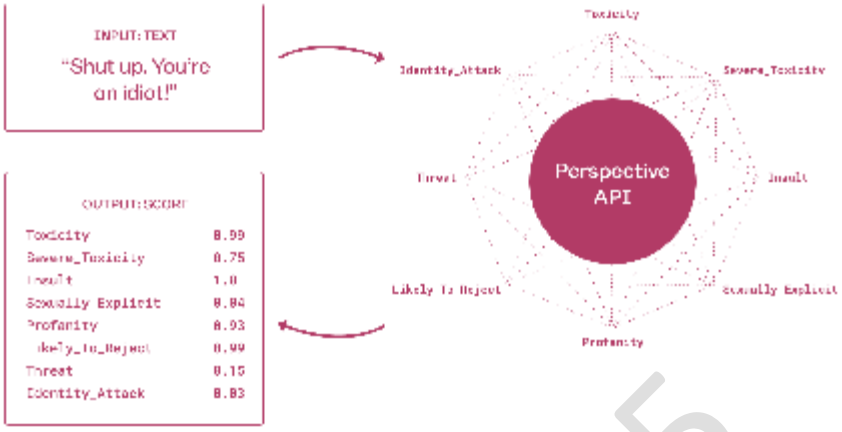


Рисунок 2.3 - Аналіз токсичності тексту Perspective API

Перспективні моделі надають бали для кількох різних атрибутів. Окрім флагманського атрибута Токсичність, ось деякі інші атрибути, за якими Perspective може надати бали:

- Сильна токсичність,
- Образа,
- Ненормативна лексика,
- Атака на ідентифікацію,
- Загроза,
- Відвертий сексуальний характер.

Amazon Comprehend використовує обробку природної мови (NLP) для отримання інформації про вміст документів. Він розвиває ідеї, розпізнаючи сутності, ключові фрази, мову, настрої та інші загальні елементи в документі.

Усі функції Amazon Comprehend приймають текстові документи UTF-8 як вхідні дані. Крім того, настроювана класифікація та настроюване розпізнавання об’єктів приймають файли зображень, PDF-файли та файли Word як вхідні дані.

Amazon Comprehend використовує попередньо навчену модель для вивчення та аналізу документа або набору документів, щоб отримати інформацію про них. Ця модель безперервно навчається на великому об'ємі тексту, тому вам не потрібно надавати навчальні дані.

Amazon Comprehend аналізує наступні типи статистики:

– Сутності – посилання на імена людей, місця, предмети та місця, які містяться в документі.

– Ключові фрази – фрази, які з'являються в документі. Наприклад, документ про баскетбольну гру може містити назви команд, назву місця проведення та остаточний рахунок.

– Особиста інформація (PII) – персональні дані, за якими можна ідентифікувати особу, наприклад адреса, номер банківського рахунку або номер телефону.

– Мова – домінуюча мова документа.

– Настрої – домінуючий настрій документа, який може бути позитивним, нейтральним, негативним або змішаним.

– Цільовий настрій – настрої, пов'язані з певними об'єктами в документі. Настрій для кожного випадку сутності може бути позитивним, негативним, нейтральним або змішаним.

– Синтаксис – частини мови для кожного слова в документі.

Google Cloud Natural Language API надає потужні інструменти для аналізу та розуміння тексту, включаючи визначення тональності та синтаксичний розбір.

Google Natural Language API — це простий у використанні інтерфейс для набору потужних моделей NLP, які були попередньо навчені Google для виконання різноманітних завдань. Оскільки ці моделі були навчені на величезних масивах документів, їхня продуктивність зазвичай досить висока, якщо вони використовуються на наборах даних, які не використовують дуже специфічну мову.

Найбільша перевага використання цих попередньо навчених моделей через API полягає в тому, що не потрібен навчальний набір даних. API дозволяє користувачеві негайно почати робити прогнози, що може бути дуже цінним у ситуаціях, коли доступно мало позначених даних.

API природної мови складається з п’яти різних служб:

– Синтаксичний аналіз

– Аналіз настроїв

– Аналіз сутностей

– Аналіз настрою сутності

– Класифікація тексту

TextRazor – це API для аналізу тексту, що надає інструменти для синтаксичного аналізу, вилучення сутностей та визначення тональності.

API TextRazor можна легко інтегрувати з будь-якою мовою, яка може надсилати HTTP-запит і аналізувати відповідь JSON, що робить можливою потужну текстову аналітику лише за допомогою кількох рядків коду. TextRazor дозволяє витягувати будь-яку необхідну інформацію в одному запиті, зв’язуючи витягнуті семантичні метадані, щоб полегшити ідентифікацію складних шаблонів.

Відмовостійка інфраструктура TextRazor побудована на основі хмари Amazon Web Services і фізичного обладнання.

DeText — це структура розуміння Deep Text для завдань ранжирування, класифікації та створення мови, пов’язаних із NLP. Він використовує семантичну відповідність за допомогою глибоких нейронних мереж, щоб зрозуміти наміри учасників у системах пошуку та рекомендацій.


Рисунок 2.4 – Логотип DeText

Як загальний фреймворк NLP, DeText можна застосовувати для багатьох завдань, включаючи пошук і ранжування рекомендацій, багатокласову класифікацію та завдання розуміння запитів.

Основні можливості DeText:

– Семантичний аналіз тексту з використанням глибоких нейронних мереж.

– Підтримка завдань пошуку, класифікації та генерації тексту.

– Використання LSTM, CNN, BERT для кодування текстових даних.

– Гнучкість у налаштуванні моделі для різних потреб.

DeText приймає текстові дані (наприклад, пошуковий запит), обробляє їх через попередньо навчені NLP-моделі та виводить відповідний результат, наприклад:

– У пошуку: підбирає релевантні результати на основі семантичної схожості.

– У рекомендаціях: аналізує текст запиту та формує персоналізовані пропозиції.

– У класифікації: визначає, до якої категорії належить текст.

Фреймворк використовується в LinkedIn для покращення пошуку та рекомендацій, підвищуючи точність системи.

NLTK (англ. Natural Language Toolkit) - це набір бібліотек і програм для символьної та статистичної обробки природної мови (NLP) для англійської мови, написаних мовою програмування Python.

NLTK призначений для підтримки досліджень і викладання навчальних курсів пов'язаних з НЛП та близькоспорідненими областями, включаючи емпіричну лінгвістику, когнітивну науку, штучний інтелект, пошук інформації та машинне навчання. NLTK успішно використовується як навчальний інструмент, а також як платформа для створення прототипів і побудови дослідницьких систем. У США та ще у 25 країнах 32 університети використовують NLTK у своїх курсах. NLTK підтримує функціональні можливості класифікації, токенізації, стемінгу, тегів, аналізу та семантичного міркування.

Завдяки практичному посібнику, який представляє основи програмування разом із темами з комп’ютерної лінгвістики, а також вичерпну документацію API, NLTK підходить як для лінгвістів, інженерів, студентів, викладачів, дослідників, так і для промислових користувачів. NLTK доступний для Windows, Mac OS X і Linux. Найкраще те, що NLTK є безкоштовним проектом із відкритим кодом, керованим спільнотою.

spaCy — бібліотека програмного забезпечення з відкритим вихідним кодом для обробки природної мови, написана на мовах програмування Python і Cython.

На відміну від NLTK, який широко використовується для навчання та досліджень, spaCy зосереджується на наданні програмного забезпечення для виробничого використання. spaCy також підтримує робочі процеси глибокого навчання, які дозволяють підключати статистичні моделі, навчені популярними бібліотеками машинного навчання, такими як TensorFlow, PyTorch або MXNet, через власну бібліотеку машинного навчання Thinc. Використовуючи Thinc як бекенд, spaCy пропонує моделі згорткових нейронних мереж для розмічування частин мови, розбору залежностей, категоризації тексту та розпізнавання

іменованих сутностей (PIC). Попередньо створені моделі нейронних мереж для виконання цього завдання доступні для 17 мов, серед яких українська поки відсутня, хоча є багатомовна модель PIC.

Додаткова підтримка токенізації для більш ніж 65 мов дозволяє користувачам також навчати власні моделі на власних наборах даних.

Flair – це бібліотека для обробки природної мови (NLP) яка розроблена Берлінським університетом Гумбольдта та друзями, заснована на глибоких нейронних мережах. Вона надає прості у використанні інтерфейси до виконання різних завдань NLP, включаючи аналіз тональності.

Flair є:

– Потужна бібліотека NLP. Flair дозволяє застосовувати до тексту наші найсучасніші моделі обробки природної мови (NLP), такі як розпізнавання іменованих об’єктів (NER), аналіз настроїв, теги частини мови (PoS), спеціальна підтримка біомедичних текстів , усунення неоднозначності та класифікація з підтримкою швидко зростаючої кількості мов.

– Бібліотека для вбудовування тексту. Flair має прості інтерфейси, які дозволяють використовувати та комбінувати різні вбудовані слова та документи, включаючи запропоновані нами вбудовані Flair та різноманітні трансформери.



## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

При розробці програмного забезпечення системи штучного інтелекту для аналізу текстів на наявність інформаційно-психологічних впливів було обрано кілька ключових технологій: середовище розробки Unity, а також мови програмування C# та Python

Для створення штучного інтелекту було використано Python. Переваги Python для машинного навчання та ШІ Серед плюсів мови Python для спеціалістів зі ШІ, ML і Data Science можна виділити:

– Простоту вивчення та використання. Зрозумілий синтаксис Python дозволяє писати чистий і читабельний код, щоб швидше виконувати завдання та економити зусилля.

– Бібліотеки Python для штучного інтелекту і машинного навчання. Python має величезну кількість бібліотек і фреймворків для створення і тренування ML-моделей.

– Гнучкість. Мова Python підтримує декілька парадигм програмування, як-от функціональне, об’єктно-орієнтоване та процедурне. А ще інтегрується з іншими мовами, що корисно для масштабних проєктів.

Для створення інтерфейсу програми було використано Unity.

Unity — багатоплатформовий інструмент для розроблення відеоігор і застосунків, і рушій, на якому вони працюють. Створені за допомогою Unity програми працюють на настільних комп'ютерних системах, мобільних пристроях та гральних консолях у дво- та тривимірній графіці, та на пристроях віртуальної чи доповненої реальності. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL.

В даному проєкті за допомогою Unity розроблено:

– Зручний інтерфейс користувача

– Візуалізовано результати аналізу текстів

– Кросплатформеність програми

Основною мовою програмування в середовищі Unity є C#. Також C# використовувався для написання алгоритмів які аналізують текст та передання даних нейронній мережі за допомогою бібліотеки Python.NET.

Python.NET ( pythonnet) — це пакет, який забезпечує програмістам Python майже повну інтеграцію з .NET Framework, .NET Core і середовищем виконання Mono у Windows, Linux і macOS. Python.NET надає потужний інструмент

створення сценаріїв додатків для розробників .NET. Використовуючи цей пакет, ви можете створювати сценарії додатків .NET або створювати цілі додатки на Python, використовуючи служби та компоненти .NET, написані будь-якою мовою, націленою на CLR (C#, VB.NET, F#, C++/CLI).

Були обрані два середовища розробки PyCharm для Python та Visual Studio.

PyCharm — інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний зневаджувач, інструмент для запуску юніт-тестів і підтримує веброзробку на Django.

Можливості PyCharm:

– Статичний аналіз коду, підсвічування синтаксису і помилок.

– Навігація серед проєктів і початкового коду: відображення файлової структури проєкту, швидкий перехід між файлами, класами і методами.

– Рефакторинг: перейменування, витяг методу, введення змінної, введення константи, підняття і опускання методу тощо.

– Інструменти для веброзробки з використанням фреймворку Django

– Вбудований зневаджувач для Python

– Вбудовані інструменти для юніт-тестування

– Розробка з використанням Google App Engine

– Підтримка систем контролю версій: загальний користувацький інтерфейс для Mercurial, Git, Subversion, Perforce і CVS з підтримкою списків змін та злиття

Microsoft Visual Studio — серія продуктів фірми Майкрософт, які містять інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці продукти дають змогу розробляти як консольні програми, так і програми з графічним інтерфейсом, включно з підтримкою технології Windows Forms, а також вебсайти, вебзастосунки, вебслужби як у рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Visual Studio включає низку передових інструментів для розробки програмного забезпечення, включаючи:

– Розумний редактор коду з IntelliSense який автоматично пропонує завершення коду.

– Розширена налагодження помилок для виявлення помилок та оптимізації коду.

– Підтримка контролю за вихідним кодом з вбудованою інтеграцією з Git та GitHub.

– Підтримка багатьох платформ, дозволяючи розробку на Windows, macOS та Linux.

Visual Studio використовується для розробки широкого спектра додатків, включаючи:

– Додатки для робочого столу Windows: завдяки підтримці для .NET, C++ та інші технології, ідеально підходить для створення бізнес-програмного забезпечення та настільних застосунків.

– Веб-сайти та веб-додатки: пропонує інтеграцію з ASP.NET, HTML, CSS та JavaScript, дозволяючи розробку динамічних сайтів та веб-базованих додатків.

– Мобільний додаток для Android та iOS: завдяки Xamarin, можна розробляти кросплатформені додатки з єдиною базою коду.

– Розробка хмарних технологій: інтеграція з Microsoft Azure дозволяє створювати, тестувати та розгортати додатки у хмарі.

– Розробка відеоігор: підтримує інструменти для розробка гри з Unity, роблячи його популярним вибором серед розробників ігор.

– Машинне навчання та штучний інтелект: завдяки інтеграції з бібліотеками штучний інтелект та автоматичне навчання, є корисним для проектів, що базуються на великих даних та ШІ.

## 2.3 Розгорнута постановка завдання

Основним призначенням і завдання програмного забезпечення, що розробляється, є виявлення інформаційно-психологічних впливів у текстах. Це передбачає, що в систему від користувача будуть надходити вхідні дані у вигляді текстової інформації англійською мовою. Обробка здійснюється за допомогою алгоритмів сентимент-аналізу, аналізу маніпулятивної мови, оцінки лексичного розмаїття та визначення рівня суб'єктивності. На основі цих характеристик нейронна мережа класифікує текст за відповідними категоріями та формує висновок.

Мова для розробки програмного забезпечення є C# та Python. Графічний інтерфейс реалізований за допомогою середовища Unity.

Користувацький інтерфейс передбачає головне меню та вікно застосунку. Головне меню містить назву додатку та три кнопки:

- Запуск аналізу – відкриває головне вікно застосунку.
- Інформація – відображає довідку про програму.
- Вихід – завершує роботу програми.

Вікно застосунку передбачає наявність поля для вводу вхідних даних, кнопку для початку аналізу даних та кнопку для очищення поля.

Також важливим є забезпечення функції збереження отриманих результатів в окремий текстовий файл за бажанням користувача для довготривалого збереження та обробки отриманих даних.

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

Система аналізу текстів на наявність інформаційно-психологічних впливів працює за принципом послідовної обробки вхідних даних, застосування алгоритмів аналізу та подальшої візуалізації результатів.

Для створення ефективної системи аналізу текстів на наявність інформаційно-психологічних впливів необхідно дотримуватись низки важливих принципів. Першим кроком є визначення основних вимог до системи, її функціональність, швидкість обробки даних і точність аналізу. Крім того, важливо зазначити, що обробка великих обсягів текстової інформації потребує оптимізації алгоритмів, які необхідні для безперебійної роботи системи навіть при високих навантаженнях.

Одним із центральних моментів є створення зручного інтерфейсу, який дозволить користувачам легко вводити дані для аналізу, переглядати результати та взаємодіяти з системою без додаткових труднощів. Очікується, що буде доступне завантаження текстів у різних форматах, що спростить роботу тим користувачам, які мають справу з великими текстовими масивами. Також система має забезпечувати збереження результатів аналізу для подальшого використання.

Функціонування системи базується на послідовному виконанні таких етапів:

1. Отримання вхідних даних.

Після запуску головного вікна програми перед користувачем буде поле де присутній текст, який описує поле. Нижче поля дві кнопки для початку аналізу та очищення тексту.

Коли текст для аналізу буде введений у поле користувач має натиснути на кнопку для виконання аналізу.

2. Попередня обробка тексту

Попередня обробка тексту є важливим етапом у процесі аналізу, оскільки дозволяє підготувати дані до подальшої обробки та зменшити кількість зайвої або нерелевантної інформації. Це сприяє підвищенню точності аналізу та ефективності роботи алгоритмів.

– Видалення зайвих символів. На початковому етапі система видаляє всі зайві символи, такі як пунктуаційні знаки (крапки, коми, знаки питання тощо), спеціальні символи (#, @, %, &, *, тощо) та зайві пробіли. Це необхідно для того, щоб уникнути впливу непотрібних символів на результати аналізу та покращити якість подальшої обробки тексту.

– Приведення тексту до нижнього регістру. Для забезпечення коректності аналізу всі слова в тексті перетворюються до нижнього регістру. Це дозволяє уникнути дублювання одного й того ж слова у різних формах написання.

3. Аналіз тексту.

Система проводить комплексну оцінку тексту, використовуючи такі алгоритми:

– Аналіз сентименту. Визначення емоційного забарвлення тексту (позитивний, негативний, нейтральний). Для цього використовуються методи на основі лексичних ресурсів, зокрема бібліотеки Vader і TextBlob.

Vader (Valence Aware Dictionary and sEntiment Reasoner) — це інструмент аналізу настроїв на основі правил, спеціально розроблений для аналізу текстів у соціальних мережах. Vader — це попередньо навчена модель аналізу настрою, яка надає оцінку настрою для заданого тексту.

Vader використовує словник слів і правила, щоб визначити почуття фрагмента тексту. Він використовує оцінку валентності для кожного слова, щоб визначити його позитивність чи негативність.

TextBlob — це бібліотека Python з відкритим кодом для обробки текстових даних. він пропонує простий API для доступу до своїх методів і виконання основних завдань NLP. TextBlob виконує різні операції з текстовими даними, як-от вилучення фраз іменників, аналіз настроїв, класифікація, переклад тощо.

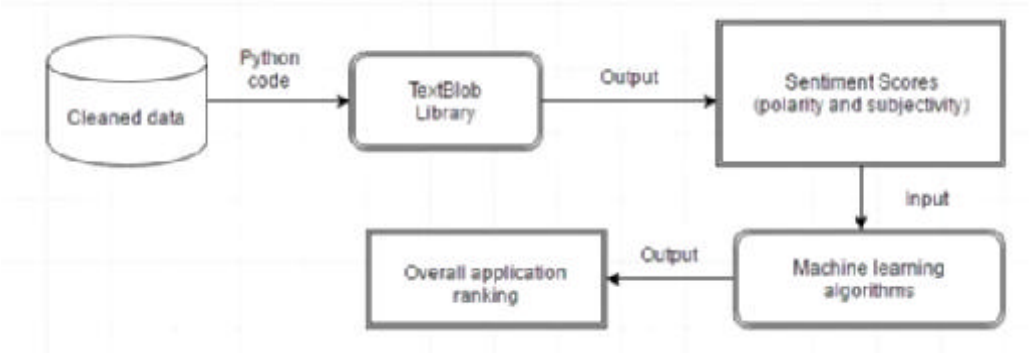


Рисунок 3.1 - Архітектура аналізу даних з використанням бібліотеки TextBlob

TextBlob створено на основі NLTK і Pattern, він дуже простий у використанні та може обробляти текст у кількох рядках коду. TextBlob може допомогти вам почати роботу з завдань NLP.

Формула розрахунку загального сентименту:

$$S = \frac{V+T}{2},$$

де V – оцінка сентименту за допомогою Vader, T – оцінка сентименту за допомогою TextBlob.

– Аналіз маніпулятивної мови. Перевірка частоти вживання слів із заздалегідь визначених маніпулятивних словників. Розрахунок відносної частоти маніпулятивних слів виконується за формулою:

$$\mathrm{M_f} = \frac{\mathrm{M}}{\mathrm{T}},$$

де M – кількість маніпулятивних слів у тексті, T – загальна кількість слів.

– Обчислення лексичного розмаїття. Використовується індекс типи-токени (TTR), що визначає співвідношення унікальних слів до загальної кількості слів у тексті:

$$\mathrm{TTR} = \frac{\mathrm{U}}{\mathrm{T}},$$

де U – кількість унікальних слів, T – загальна кількість слів.

– Оцінка рівня суб'єктивності. Визначення співвідношення об'єктивної та суб'єктивної інформації в тексті. Для розрахунку данного аналізу використовувалось Вираховується через співвідношення кількості суб'єктивних слів до загальної кількості слів:

$$S_r = \frac{S_u}{T},$$

де $S_u$ – кількість суб'єктивних слів, T – загальна кількість слів.

4. Передача даних між компонентами.

Отримані результати аналізу передаються до нейронної мережі для подальшої класифікації за допомогою бібліотеки Python.NET.

Нейронна мережа отримує на вхід такі параметри:

- Загальний сентимент тексту.
- Аналіз маніпулятивної мови
- Індекс лексичного розмаїття (TTR).
- Рівень суб'єктивності тексту.

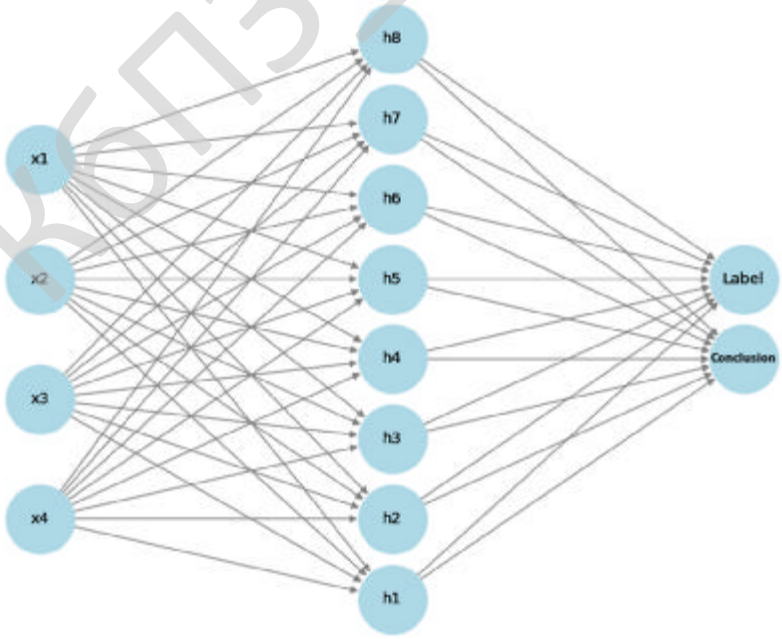


Рисунок 3.2 – Структура нейронної мережі

Структура нейронної мережі:

– Вхідний шар: 4 нейрони (по одному на кожен параметр).

– Прихований шар: складається з 8 нейронів, які застосовують активаційну функцію ReLU. Цей шар вивчає нелінійні залежності між параметрами тексту для прийняття рішень.

– Вихідний шар: 2 нейрони, що відповідають за класифікацію тексту(Label) та генерацію підсумкового висновку на основі попередніх результатів(Conclusion), функція активації Softmax.

Нейронна мережа навчається на основі набору даних з позначеними прикладами текстів, використовуючи алгоритм оптимізації Adam та функцію втрат sparse_categorical_crossentropy.

5. Формування висновку. На основі вихідних значень нейромережі система приймає рішення щодо рівня інформаційно-психологічного впливу тексту. Результати відображаються у вигляді текстового звіту, що дозволяють користувачеві оцінити характеристики тексту.

6. Функціональні можливості для користувача. Система передбачає можливість:

– Очищення введених даних.

– Збереження результатів у текстовий файл.

– Ознайомлення з інструкціями щодо роботи із системою.



### 3.2 Розробка структурної схеми

Структурною називають схему, яка відображає склад і взаємодію з управління частин розроблюваного програмного забезпечення. Для програмного забезпечення системи штучного інтелекту для аналізу текстів на наявність інформаційно-психологічних впливів, структурна схема має вигляд як на рисунку 3.3.

Графічний інтерфейс Unity забезпечує введення тексту. Користувач вводить текст, який потрібно проаналізувати. Після введення текст передається

на обробку вхідних даних.

Обробка вхідних даних включає очищення тексту від спеціальних символів, зайвих пробілів та неприпустимих знаків. Після цього текст поділяється на речення та слова, що дозволяє точніше провести аналіз.

Система аналізу тексту проводить чотири типи аналізу: лексичне розмаїття, аналіз маніпулятивної мови, сентимент-аналіз та суб'єктивність. Лексичне розмаїття, аналіз маніпулятивної мови та суб'єктивність мають діапазон від 0 до 1. Аналіз маніпулятивної мови оцінює рівень впливу тексту в цьому діапазоні, визначаючи, наскільки текст містить маніпулятивні слова чи вирази. Сентимент-аналіз має діапазон від -1 до 1, де -1 означає негативне забарвлення тексту, 0 — нейтральне, а 1 — позитивне. Лексичне розмаїття визначає різноманітність використовуваних слів, що допомагає оцінити, наскільки текст багатий на лексичні варіації. Суб'єктивність показує, наскільки текст базується на суб'єктивних судженнях, тобто наскільки в ньому присутні елементи, що виражають особисті погляди або оцінки.

Нормалізація даних виконує приведення всіх вхідних значень до єдиного масштабу, що дозволяє усунути нерівномірність у розподілі параметрів. Вона гарантує, що всі значення аналізів знаходяться в однаковому діапазоні, що дозволяє нейромережі коректно їх обробляти. Крім цього, нормалізація передбачає перетворення текстових міток у числовий формат для забезпечення коректної роботи системи. Це дозволяє однаково обробляти всі категорії даних без втрати їхньої значущості.

Завдяки нормалізації забезпечується коректне навчання та функціонування нейромережі, що дозволяє отримувати точні результати аналізу тексту.

Нейромережа складається з трьох рівнів: вхідного, прихованого та вихідного. На вхідному шарі — чотири нейрони, що відповідають за лексичне розмаїття, маніпулятивну мову, сентимент-аналіз та суб'єктивність. Прихований шар складається з восьми нейронів, які виконують більш складну обробку даних, отриманих з вхідного шару, і дозволяють створити абстрактне

уявлення про текст. Вихідний шар містить два нейрони: один відповідає за класифікацію тексту, інший — формує підсумковий висновок.

Графічний інтерфейс відображає результати аналізу у зрозумілому форматі. Користувач отримує числові значення кожного параметра аналізу. Класифікація тексту визначає його тип, а висновок узагальнює результати аналізу. Відображаються такі параметри, як сентимент аналіз, лексичне розмаїття, аналіз маніпулятивної мови та суб'єктивність. Після завершення користувач може зберегти результати у текстовий файл.

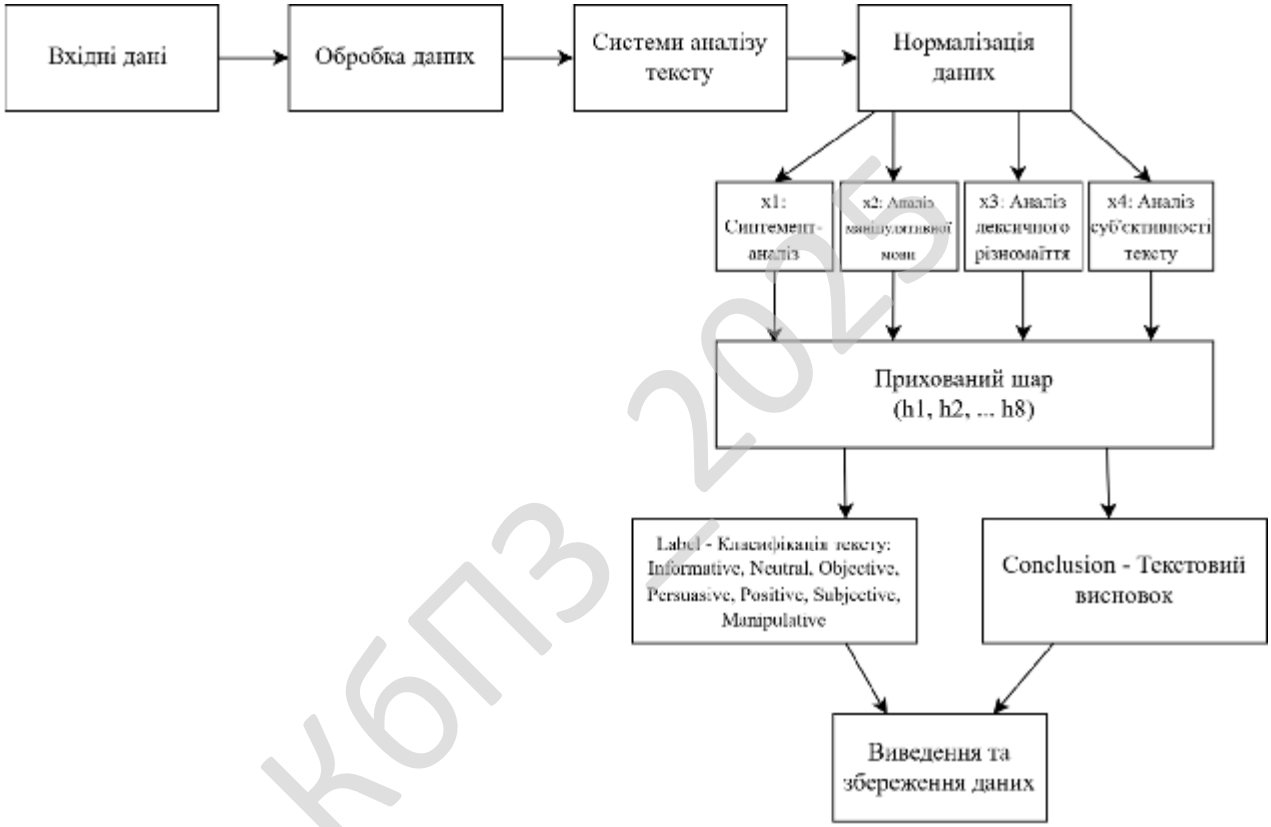


Рисунок 3.3 – Структурна схема

### 3.3 Розробка функціональної схеми

Функціональна схема або схема даних - схема взаємодії компонентів програмного забезпечення з описом інформаційних потоків, складу даних у потоках і вказівкою використовуваних файлів і пристроїв. На рисунку 3.4 функціональна схема системи демонструє логіку роботи системи штучного

інтелекту для аналізу текстів на наявність інформаційно-психологічних впливів

Функціонально вся система розбита на 3 логічні блоки: нейрона мережа, інтерфейс користувача та обробка тексту.

Інтерфейс користувача – це графічний інтерфейс, створений за допомогою середовища Unity, який дозволяє користувачеві вводити текст для аналізу та отримувати результати класифікації інформаційних впливів. Цей модуль також включає в себе збереження результатів дослідження в окремий файл.

Обробка тексту включає в себе декілька модулів:

– Модуль очищення тексту – видаляє всі небажані символи, такі як розділові знаки та спеціальні символи, залишаючи тільки літери, цифри, пробіли та дефіси. Після цього текст розбивається на окремі слова для подальшої обробки.

– Модуль сентимент-аналізу – визначає емоційне забарвлення тексту (позитивне, негативне або нейтральне). Модуль реалізовано за допомогою TextBlob та VADER для більш точного аналізу емоційного контексту в текстах.

– Модуль аналізу маніпулятивної мови – виявляє слова та вирази, що можуть впливати на свідомість читача.

– Модуль оцінки лексичного розмаїття – аналізує різноманітність словника, визначаючи рівень повторюваності слів.

– Модуль визначення рівня суб'єктивності – оцінює наявність суб'єктивних висловлювань у тексті. Реалізовано за допомогою TextBlob для визначення рівня суб'єктивності.

Результати обробки тексту за допомогою бібліотеки Python.NET передаються в нейромережу. Нейронна мережа для аналізу створена з використанням TensorFlow і має такі модулі:

– Модуль нормалізації даних. Цей модуль приводить вхідний текст до єдиного формату для коректного аналізу. Основні функції: усуває нерівномірності у розподілі значень, виконує масштабування за допомогою MinMaxScaler і кодування лейблів через LabelEncoder, який перетворює

категоріальні значення у числові.

– Модуль класифікації тексту. Модуль аналізує отримані метрики та визначає категорію тексту.

– Модуль формування висновку. Формує підсумковий висновок на основі результатів аналізу

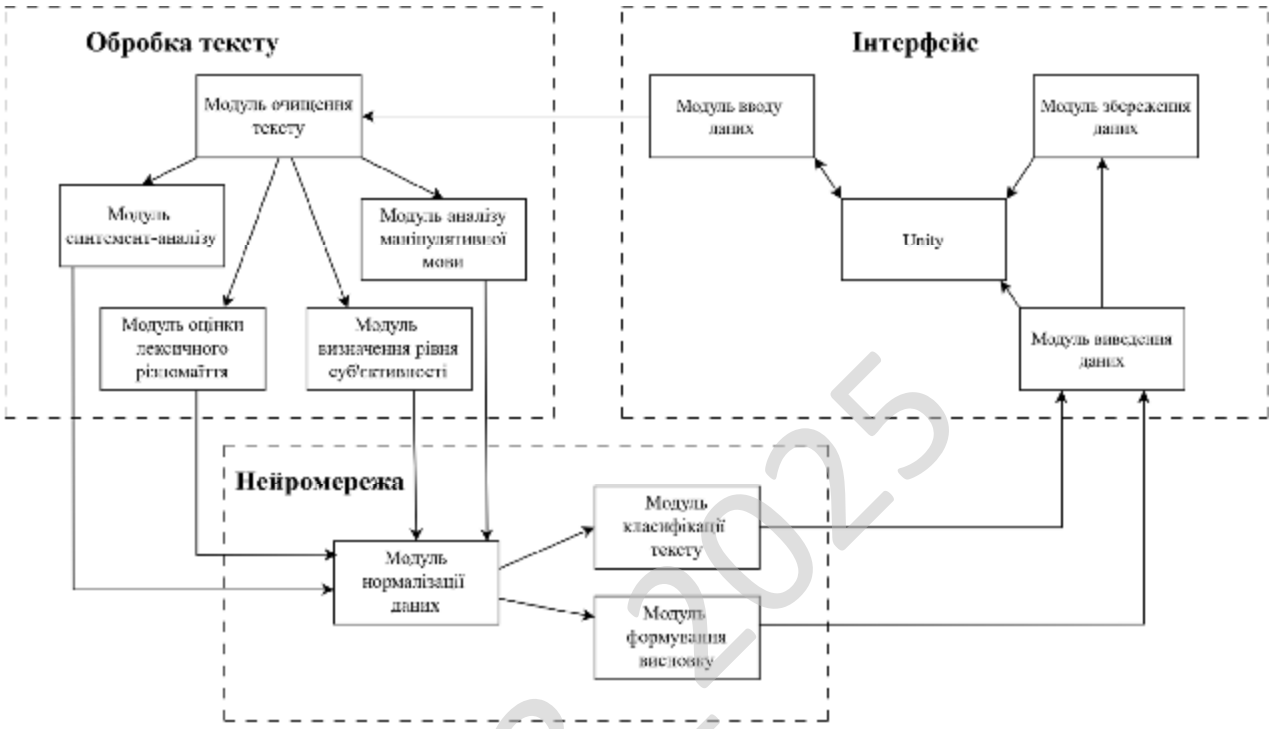


Рисунок 3.4 - Функціональна схема системи

### 3.4 Розробка діаграми процесів

Діаграма процесів у ПЗ — це графічне представлення послідовності дій, які виконуються в програмі для досягнення певного результату. Вона показує, як обробляються дані, які етапи проходять, та як взаємодіють різні компоненти системи.

Діаграма процесів, яка показує логіку системи та потік даних в ній, зображена на рисунку 3.5.

Після запуску програмного забезпечення з’являється головне вікно застосунку системи штучного інтелекту для аналізу текстів на наявність інформаційно-психологічних впливів. Після введення та перевірки даних

вхідного тексту розпочинається аналіз тексту відповідними алгоритмами. Далі результати аналізів нормалізуються та передаються нейронній мережі. Потім нейрона мережа на основі отриманих даних класифікує текст та робить висновок щодо тексту. Після цього результати аналізу виводяться на екран, також результати можна зберегти в текстовий файл.

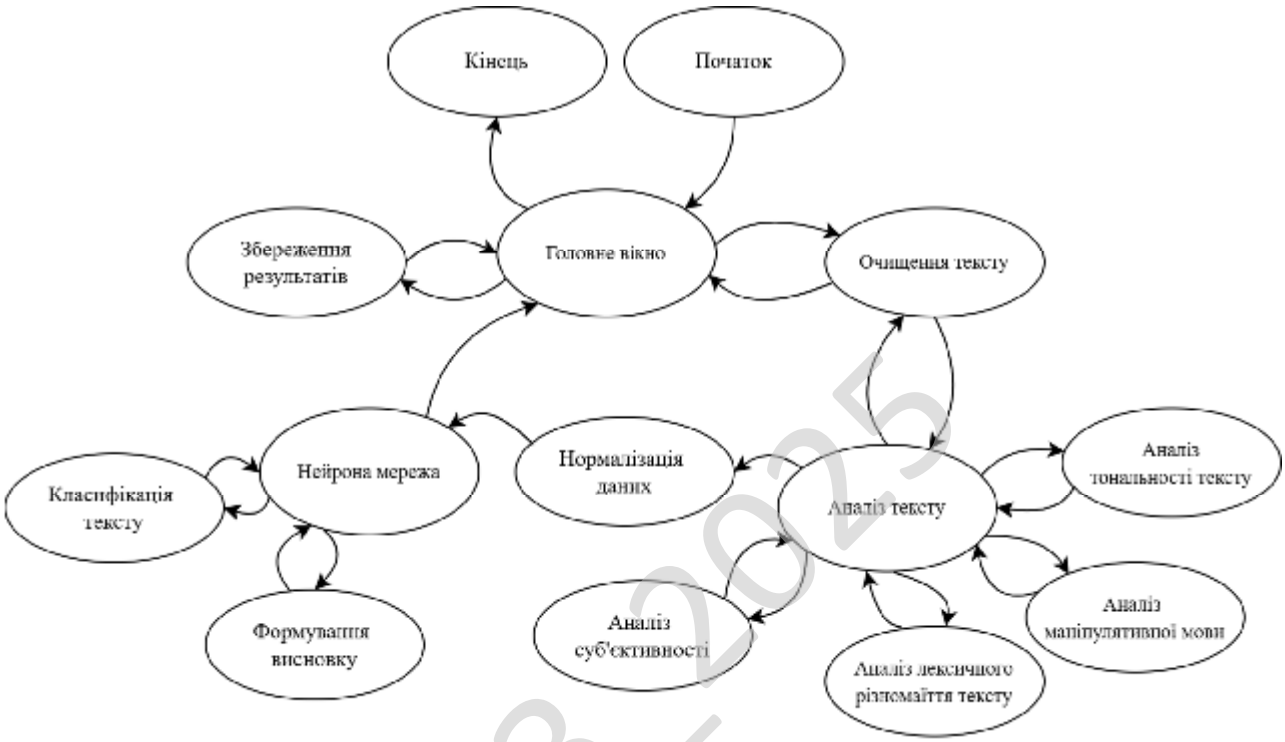


Рисунок 3.5 – Діаграма процесів

# 4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

## 4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з таких пунктів:

– Виведення головного вікна

– Введення текстових даних

– Перевірка необхідності очищення тексту. Якщо так – очистка даних і повернення до введення, якщо ні – перехід до аналізу тексту

– Аналіз тексту

– Передача даних нейронній мережі

– Отримання результатів від нейромережі

– Виведення та збереження результатів

На рисунку 4.2 наведено блок-схему підпрограми нейромережі. Її робота складається з таких пунктів:

– Отримання вхідних даних

– Нормалізація даних

– Перевірка наявності моделі. Якщо модель відсутня – навчання моделі, якщо модель є - передача вхідних даних у модель нейромережі.

– Передача вхідних даних у модель нейромережі

– Обчислення результату

– Повернення результатів в основну програму

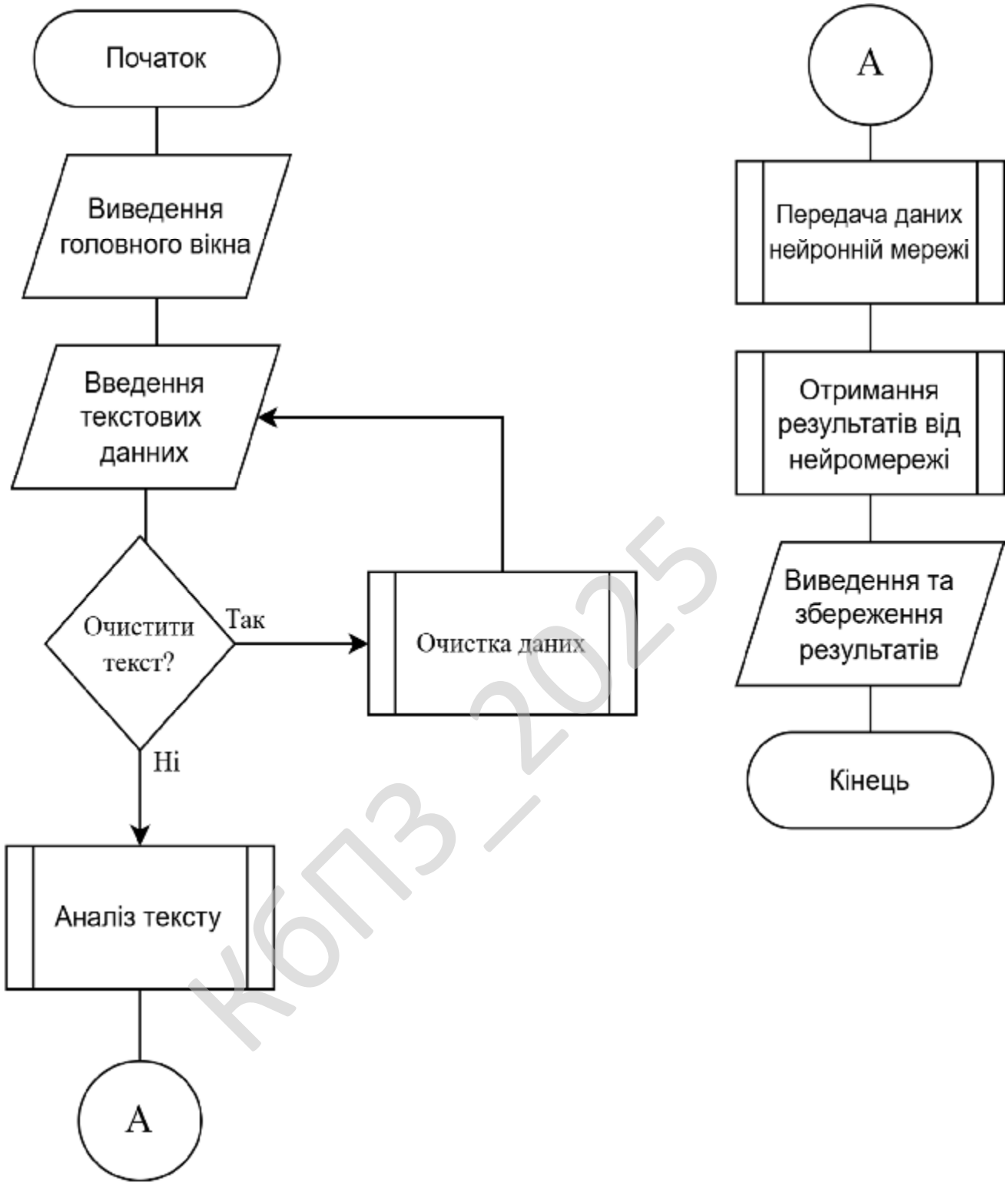




Рисунок 4.1 - Блок-схема основної програми

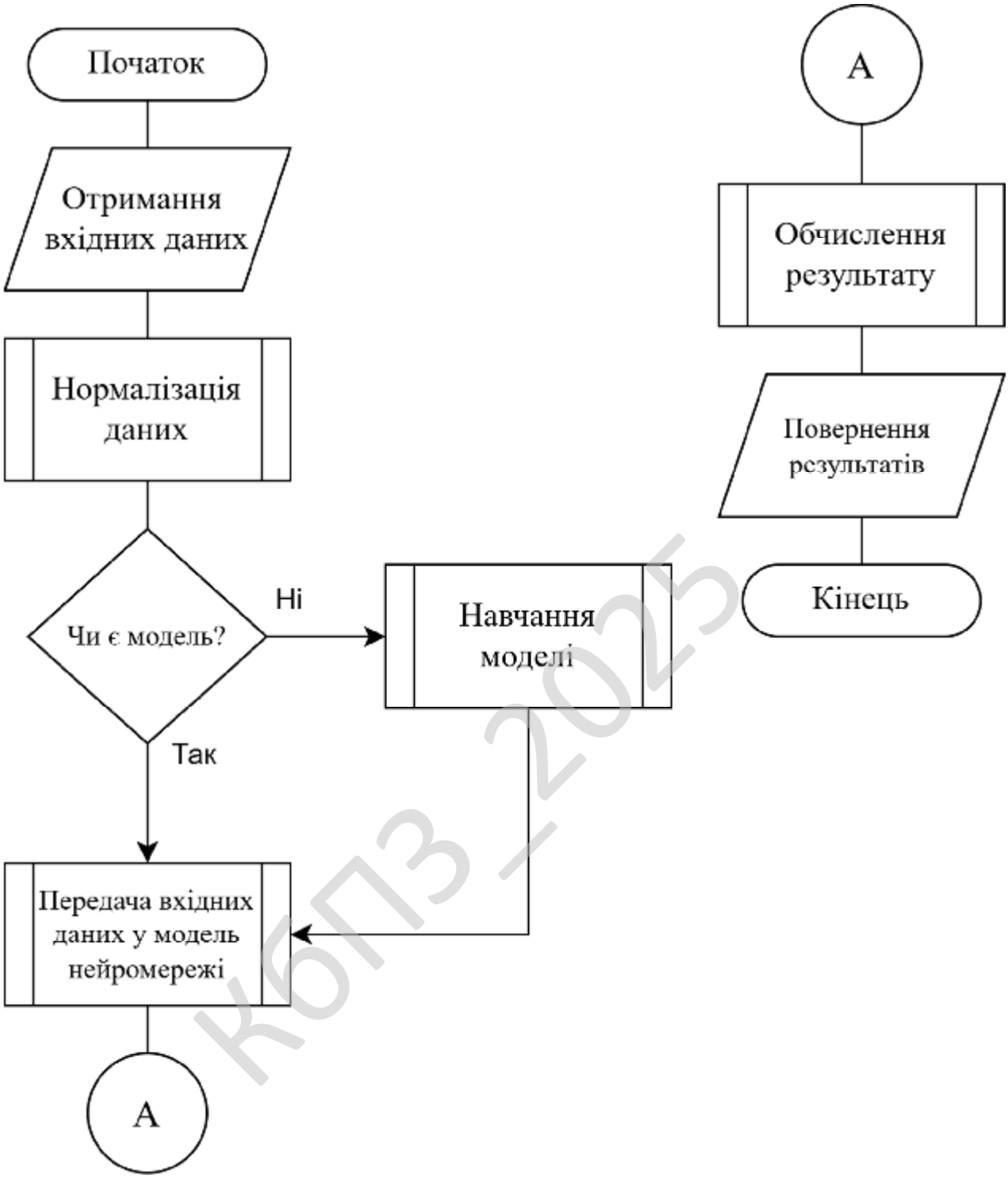


Рисунок 4.2 – Блок-схема підпрограми нейронної мережі

**Опис алгоритмів функціонування системи**

Для виявлення інформаційного впливу в текстах було створено 4 алгоритми: аналіз сентименту, частотний аналіз маніпулятивних слів, обчислення лексичного розмаїття та оцінка рівня суб'єктивності. На основі

результатів цих алгоритмів нейронна мережа синтезує отримані дані та робить висновок про характер інформаційного впливу тексту.

Ана́ліз тона́льності те́ксту (сентимент-аналіз, англ. Sentiment analysis, англ. Opinion mining) — клас методів контент-аналізу в комп'ютерній лінгвістиці, призначений для автоматизованого виявлення в текстах емоційно забарвленої лексики і емоційної оцінки авторівщодо об'єктів, мова про які йде в тексті. Для створення алгоритму використовувались бібліотеки TextBlob та VADER.

TextBlob — це бібліотека Python для обробки природної мови (NLP). TextBlob активно використовувала Natural Language ToolKit (NLTK) для виконання своїх завдань. TextBlob використовує лексикон полярностей, де кожному слову присвоєно числове значення, що вказує на його полярність (від -1 до 1). Наприклад, слова з негативною конотацією отримують від'ємні значення, а позитивні слова — додатні. Кожне слово в тексті порівнюється з лексиконом, щоб визначити його полярність. Оцінки полярності всіх слів сумуються та нормалізуються, що дозволяє отримати загальну полярність тексту. Але TextBlob не завжди добре працює з короткими текстами або сленговими виразами, що може впливати на точність аналізу.

VADER — лексикон і інструмент аналізу настроїв на основі правил, спеціально розроблений для текстів, які містять неформальну мову, як-от дописи в соціальних мережах і огляди. Визначаючи зміст певного фрагмента тексту, VADER спочатку розбиває текст на окремі слова. Потім він призначає оцінку кожному слову, щоб визначити, позитивне воно чи негативне. На основі цих встановлених балів VADER остаточно обчислює загальну оцінку настрою тексту.

Аналіз маніпулятивної мови в текстах є важливою складовою досліджень у галузі лінгвістики, медіазнавства та комп'ютерної лінгвістики. Цей аналіз спрямований на виявлення та розуміння способів, якими мова використовується для прихованого впливу на свідомість та поведінку аудиторії. Даний алгоритм орієнтований на виявлення маніпулятивних фраз та слів. Для

цього був розроблений словник що включає такі категорії маніпулятивних висловлювань:

– Узагальнення.
– Заклик до дії.
– Емоційні тригери.
– На основі страху.
– Маніпулятивні контрасти.
– Помилкові аналогії.
– Підміна понять.
– Заклик до патріотизму.
– Сумнів і невпевненість.
– Терміновість і тривога.

Алгоритм проводить порівняння тексту зі списком маніпулятивних слів і фраз та обчислює співвідношення маніпулятивних слів до загальної кількості слів у тексті.

Типово-лексичне співвідношення (Type-Token Ratio, TTR) є показником лексичного розмаїття тексту, що відображає співвідношення кількості унікальних слів (types) до загальної кількості слів (tokens) у тексті. Вищий TTR свідчить про більшу різноманітність лексики, тоді як нижчий TTR може вказувати на повторюваність лексичних одиниць. Високий TTR може свідчити про використання різноманітної лексики, що може бути характерним для текстів, спрямованих на вплив на аудиторію через багатство виражальних засобів.

Алгоритм визначає, скільки унікальних слів міститься в тексті, порівняно з загальною кількістю слів. Спочатку з тексту видаляються всі зайві символи, а потім текст розбивається на слова. Унікальні слова збираються в множину, яка автоматично виключає повтори. Лексичне різноманіття обчислюється як відношення кількості унікальних слів до загальної кількості слів у тексті.

Аналіз суб’єктивності досліджує ставлення, почуття та висловлені думки щодо продуктів, послуг, тем або проблем. Як основне завдання, він класифікує текст як суб'єктивний або об'єктивний. У той час як суб’єктивний текст висловлює думку про об’єкт або проблему за допомогою виразів настроїв, об’єктивний текст описує об’єкт або проблему з урахуванням їх фактів. Суб'єктивність вимірюється в межах від 0 до 1, де 0 вказує на об'єктивний текст, а 1 — на дуже суб'єктивний.

Для створення алгоритму було використано бібліотеку TextBlob. Для кожного речення визначається його суб'єктивність через функції TextBlob, що повертає значення між 0 і 1.

Далі данні передаються нейронній мережі у вигляді JSON. Ці дані нормалізуються перед тим, як подавати їх на вхід мережі. Після цього нейронна мережа обробляє ці вхідні дані, виявляючи закономірності між показниками. На основі цього вона визначає ймовірність належності тексту до певного класу і робить висновок відповідно до вхідних характеристик.

Алгоритм аналізу тональності тексту:

```
using System;
using System.IO;
using System.Text.RegularExpressions;
using Python.Runtime;
using UnityEngine;

namespace AnalysisScripts
{
    public class SentimentAnalyzer
    {
        private PyObject textBlob;

        public SentimentAnalyzer()
        {
            try
            {
                string pythonHome =
Path.Combine(Application.streamingAssetsPath, "Python");
                string pythonLib = Path.Combine(pythonHome, "Lib");
                string pythonSitePackages = Path.Combine(pythonLib, "site-
packages");

                Environment.SetEnvironmentVariable("PYTHONHOME", pythonHome,
EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONPATH", pythonLib +
";" + pythonSitePackages, EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONNET_PYDLL",
Path.Combine(pythonHome, "python311.dll"), EnvironmentVariableTarget.Process);
```

```
                    PythonEngine.Initialize();
                    Debug.Log("Python Engine initialized.");

                    using (Py.GIL())
                    {
                        PyObject sys = Py.Import("sys");
                        PyObject sysPath = sys.GetAttr("path");
                        sysPath.InvokeMethod("append", new
PyString(pythonSitePackages));
                        Debug.Log($"Added path: {pythonSitePackages}");

                        textBlob = Py.Import("textblob");
                    }
                }
                catch (Exception ex)
                {
                    Debug.LogError($"Initialization error Python:
{ex.Message}");
                }
            }

            public (double FinalSentiment, double VaderScore, double
TextBlobScore) CalculateSentimentScore(string text, double manipulativeRatio)
            {
                text = Regex.Replace(text, @"[^\w\s.!?]", "").ToLower().Trim();

                if (string.IsNullOrWhiteSpace(text))
                {
                    Debug.LogError("Input text is empty!");
                    return (0.0, 0.0, 0.0);
                }

                double totalVader = 0.0;
                double totalTextBlob = 0.0;
                int sentenceCount = 0;

                string[] sentences = text.Split(new char[] { '.', '!', '?' },
StringSplitOptions.RemoveEmptyEntries);

                using (Py.GIL())
                {
                    PyObject vader = Py.Import("vaderSentiment.vaderSentiment");
                    PyObject SentimentIntensityAnalyzer =
vader.GetAttr("SentimentIntensityAnalyzer");
                    PyObject analyzer = SentimentIntensityAnalyzer.Invoke();
                    PyObject TextBlob = textBlob.GetAttr("TextBlob");

                    foreach (string sentence in sentences)
                    {
                        if (sentence.Trim().Length > 1)
                        {
                            PyObject scores =
analyzer.InvokeMethod("polarity_scores", new PyString(sentence));
                            double compound =
scores.GetItem("compound").As<double>();
                            totalVader += compound;

                            PyObject blob = TextBlob.Invoke(new
PyString(sentence));
                            PyObject sentiment = blob.GetAttr("sentiment");
                            double textBlobPolarity =
sentiment.GetAttr("polarity").As<double>();
```

```
                            textBlobPolarity = double.IsNaN(textBlobPolarity) ?
0.0 : textBlobPolarity;

                            totalTextBlob += textBlobPolarity;
                            sentenceCount++;
                        }
                    }
                }

                double vaderScore = sentenceCount > 0 ? totalVader /
sentenceCount : 0.0;
                double textBlobScore = sentenceCount > 0 ? totalTextBlob /
sentenceCount : 0.0;

                vaderScore = Math.Max(-1, Math.Min(vaderScore, 1));
                textBlobScore = Math.Max(-1, Math.Min(textBlobScore, 1));

                double finalSentiment = (vaderScore + textBlobScore) / 2;

                double penalty = manipulativeRatio > 0.5 ? (manipulativeRatio >
0.9 ? 0.15 : manipulativeRatio > 0.7 ? 0.1 : 0.05) : 0;
                finalSentiment -= penalty;

                finalSentiment = Math.Max(-1, Math.Min(finalSentiment, 1));

                Debug.Log($"Final Sentiment: {finalSentiment}, Vader:
{vaderScore}, TextBlob: {textBlobScore}, ManipulativePenalty: -{penalty}");

                return (Math.Round(finalSentiment, 2), Math.Round(vaderScore,
2), Math.Round(textBlobScore, 2));
            }
        }
    }
```

Алгоритм аналізу маніпулятивної мови:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using UnityEngine;
using Newtonsoft.Json;

namespace AnalysisScripts
{
    public class ManipulativeWordAnalysis : MonoBehaviour
    {
        private static HashSet<string> manipulativeWords = new
HashSet<string>();
        private static List<string> manipulativePhrases = new
List<string>();

        private string filePath;

        async void Start()
        {
            filePath = Path.Combine(Application.dataPath,
"Resources/manipulative_words.json");
            await LoadManipulativeWordsAsync();
        }
```

```
        private async Task LoadManipulativeWordsAsync()
        {
            if (!File.Exists(filePath))
            {
                return;
            }

            try
            {
                string jsonContent = await
File.ReadAllTextAsync(filePath);
                var wordData =
JsonConvert.DeserializeObject<ManipulativeWordsWrapper>(jsonContent);

                manipulativeWords.Clear();
                manipulativePhrases.Clear();

                AddWordsFromCategory(wordData.generalization);
                AddWordsFromCategory(wordData.call_to_action);
                AddWordsFromCategory(wordData.emotional_triggers);
                AddWordsFromCategory(wordData.fear_based);
                AddWordsFromCategory(wordData.manipulative_contrasts);
                AddWordsFromCategory(wordData.false_analogies);
                AddWordsFromCategory(wordData.concept_substitution);
                AddWordsFromCategory(wordData.appeal_to_patriotism);
                AddWordsFromCategory(wordData.doubt_and_uncertainty);
                AddWordsFromCategory(wordData.urgency_and_alarmism);

                Debug.Log($"Loaded: {manipulativeWords.Count} words,
{manipulativePhrases.Count} phrases.");
            }
            catch (Exception ex)
            {
                Debug.LogError($"Error loading manipulative words:
{ex.Message}");
            }
        }

        private void AddWordsFromCategory(List<string> words)
        {
            if (words == null) return;

            foreach (var phrase in words)
            {
                if (phrase.Contains(" "))
                    manipulativePhrases.Add(phrase.ToLower());
                else
                    manipulativeWords.Add(phrase.ToLower());
            }
        }

        public static double CalculateManipulativeWordRatio(string text)
        {
            if (manipulativeWords.Count == 0 && manipulativePhrases.Count
== 0)
            {
                return 0;
            }

            string cleanedText = Regex.Replace(text, @"[^\w\s-]", "");
            string[] words = cleanedText.Split(new char[] { ' ', '\n',
'\r', '\t' }, StringSplitOptions.RemoveEmptyEntries);
```

```
            int manipulativeCount = 0;
            HashSet<string> countedWords = new HashSet<string>();
            HashSet<string> countedPhrases = new HashSet<string>();

            foreach (var word in words)
            {
                if (word.Length > 3 && manipulativeWords.Contains(word) &&
!countedWords.Contains(word))
                {
                    manipulativeCount++;
                    countedWords.Add(word);
                }
            }

            foreach (var phrase in manipulativePhrases)
            {
                string[] phraseWords = phrase.Split(' ');

                int matchCount = phraseWords.Count(w =>
words.Contains(w));
                double matchRatio = (double)matchCount /
phraseWords.Length;

                if (matchRatio >= 0.65 &&
!countedPhrases.Contains(phrase))
                {
                    manipulativeCount++;
                    countedPhrases.Add(phrase);
                }
            }

            Debug.Log($"Total words: {words.Length}, Unique manipulative
words/phrases found: {manipulativeCount}");

            return words.Length > 0 ? Math.Round((double)manipulativeCount
/ words.Length, 2) : 0;
        }
    }

    [Serializable]
    public class ManipulativeWordsWrapper
    {
        public List<string> generalization;
        public List<string> call_to_action;
        public List<string> emotional_triggers;
        public List<string> fear_based;
        public List<string> manipulative_contrasts;
        public List<string> false_analogies;
        public List<string> concept_substitution;
        public List<string> appeal_to_patriotism;
        public List<string> doubt_and_uncertainty;
        public List<string> urgency_and_alarmism;
    }
}
```

Алгоритм аналізу лексичного різноманіття тексту:

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using UnityEngine;
```

```
namespace AnalysisScripts
{
    public class LexicalDiversityAnalysis : MonoBehaviour
    {
        public static double CalculateLexicalDiversity(string text)
        {
            text = Regex.Replace(text, @"[^\w\s]", "").ToLower();

            string[] words = text.Split(new char[] { ' ', '\n', '\r', '\t'
}, StringSplitOptions.RemoveEmptyEntries);
            HashSet<string> uniqueWords = new HashSet<string>(words);

            if (words.Length > 0)
            {
                return Math.Round((double)uniqueWords.Count /
words.Length, 2);
            }
            else
            {
                return 0;
            }
        }
    }
}
```

Алгоритм аналізу тексту:

```
using System;
using System.IO;
using System.Text.RegularExpressions;
using Python.Runtime;
using UnityEngine;

namespace AnalysisScripts
{
    public class SubjectivityAnalyzer
    {
        private PyObject textBlob;

        public SubjectivityAnalyzer()
        {
            try
            {
                string pythonHome =
Path.Combine(Application.streamingAssetsPath, "Python");
                string pythonLib = Path.Combine(pythonHome, "Lib");
                string pythonSitePackages = Path.Combine(pythonLib, "site-
packages");

                Environment.SetEnvironmentVariable("PYTHONHOME", pythonHome,
EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONPATH", pythonLib +
";" + pythonSitePackages, EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONNET_PYDLL",
Path.Combine(pythonHome, "python311.dll"), EnvironmentVariableTarget.Process);

                PythonEngine.Initialize();
                Debug.Log("Python Engine initialized.");

                using (Py.GIL())
                {
                    PyObject sys = Py.Import("sys");
                    PyObject sysPath = sys.GetAttr("path");
```

```
                        sysPath.InvokeMethod("append", new
PyString(pythonSitePackages));
                        Debug.Log($"Added path: {pythonSitePackages}");

                        textBlob = Py.Import("textblob");
                    }
                }
                catch (Exception ex)
                {
                    Debug.LogError($"Initialization error Python:
{ex.Message}");
                }
            }

            public double CalculateSubjectivity(string text)
            {
                text = Regex.Replace(text, @"[^\w\s.!?]", "").ToLower().Trim();

                if (string.IsNullOrWhiteSpace(text))
                {
                    Debug.LogError("Input text is empty!");
                    return 0.0;
                }

                double totalSubjectivity = 0.0;
                int sentenceCount = 0;

                string[] sentences = text.Split(new char[] { '.', '!', '?' },
StringSplitOptions.RemoveEmptyEntries);

                using (Py.GIL())
                {
                    PyObject TextBlob = textBlob.GetAttr("TextBlob");

                    foreach (string sentence in sentences)
                    {
                        if (sentence.Trim().Length > 1)
                        {
                            PyObject blob = TextBlob.Invoke(new
PyString(sentence));
                            PyObject sentiment = blob.GetAttr("sentiment");
                            double subjectivity =
sentiment.GetAttr("subjectivity").As<double>();

                            subjectivity = double.IsNaN(subjectivity) ? 0.0 :
subjectivity;

                            totalSubjectivity += subjectivity;
                            sentenceCount++;
                        }
                    }
                }

                double finalSubjectivity = sentenceCount > 0 ? totalSubjectivity
/ sentenceCount : 0.0;
                finalSubjectivity = Math.Max(0, Math.Min(finalSubjectivity, 1));

                Debug.Log($"Subjectivity Score: {finalSubjectivity}");

                return Math.Round(finalSubjectivity, 2);
            }
        }
```

```
}
```

### 4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення є відкритим (open-source) і розповсюджується за ліцензією Creative Commons CC BY-SA 4.0. Це означає, що будь-який користувач може вільно використовувати, змінювати та поширювати його, за умови дотримання вимог ліцензії.

Захист від несанкціонованих змін. Для забезпечення стабільності та безпеки програмного забезпечення реалізовано такі механізми:

– Контроль офіційного репозиторію. Основний код зберігається в репозиторії (GitHub/GitLab), де внесення змін здійснюється через Pull Requests з обов'язковою перевіркою.

– Використання підписаних релізів. GPG (GNU Privacy Guard) — це система шифрування та цифрового підпису, яка дозволяє перевіряти справжність джерела коду. Використання GPG-ключів дає змогу підтвердити, що певний коміт або реліз походить від офіційного розробника.

– Захист від небажаних змін. Впроваджено механізми обмеження прав доступу, щоб лише перевірені розробники могли редагувати критичні компоненти коду.

Захист від зловживання. Оскільки програмне забезпечення є відкритим, важливо дотримуватися ліцензійних умов:

– Код не може бути використаний у комерційних цілях без відповідного дозволу (якщо використовується ліцензія CC BY-NC).

– При розповсюдженні зміненої версії необхідно вказувати авторів оригінального проєкту (за умовами CC BY-SA).

– Не допускається використання коду для створення шкідливих програм або маніпулятивних алгоритмів.

– Використання програмного забезпечення у дослідницьких цілях можливе лише при дотриманні умов ліцензії та етичних норм.

– Всі модифікації, зроблені користувачами, повинні мати відповідну документацію та бути доступними для перевірки іншими учасниками спільноти.

Запобігання кіберзагрозам. Оскільки програмне забезпечення працює з текстовими даними, важливим аспектом захисту є:

– Фільтрація вхідних даних для запобігання можливим атакам (наприклад, впровадженню шкідливих скриптів).

– Моніторинг активності у відкритому репозиторії для виявлення можливих загроз, таких як шкідливі коміти або зміни в алгоритмах аналізу даних.

Забезпечення доступності та резервне копіювання. Для запобігання втрати даних реалізовано такі заходи:

– Резервне копіювання репозиторію на кількох платформах для захисту від втрати основного сховища коду.

– Документація для користувачів і розробників, яка пояснює принципи роботи, правила внесення змін та особливості алгоритмів.

– Тестування на стабільність. Перед публікацією нових версій проходять тестування на сумісність із попередніми версіями для забезпечення безперебійної роботи системи.

Завдяки відкритому коду та прозорості розробки, система може бути покращена спільнотою, що сприяє її безпеці та ефективності. Це дозволяє швидко виявляти потенційні загрози, усувати їх і підтримувати високу якість програмного забезпечення.

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 показано головне меню програми. З нього видно, що інтерфейс головного меню складається з таких блоків:

– Назва програмного забезпечення.

– Кнопка початку аналізу.

– Кнопка про програму.

– Кнопка закриття програми.

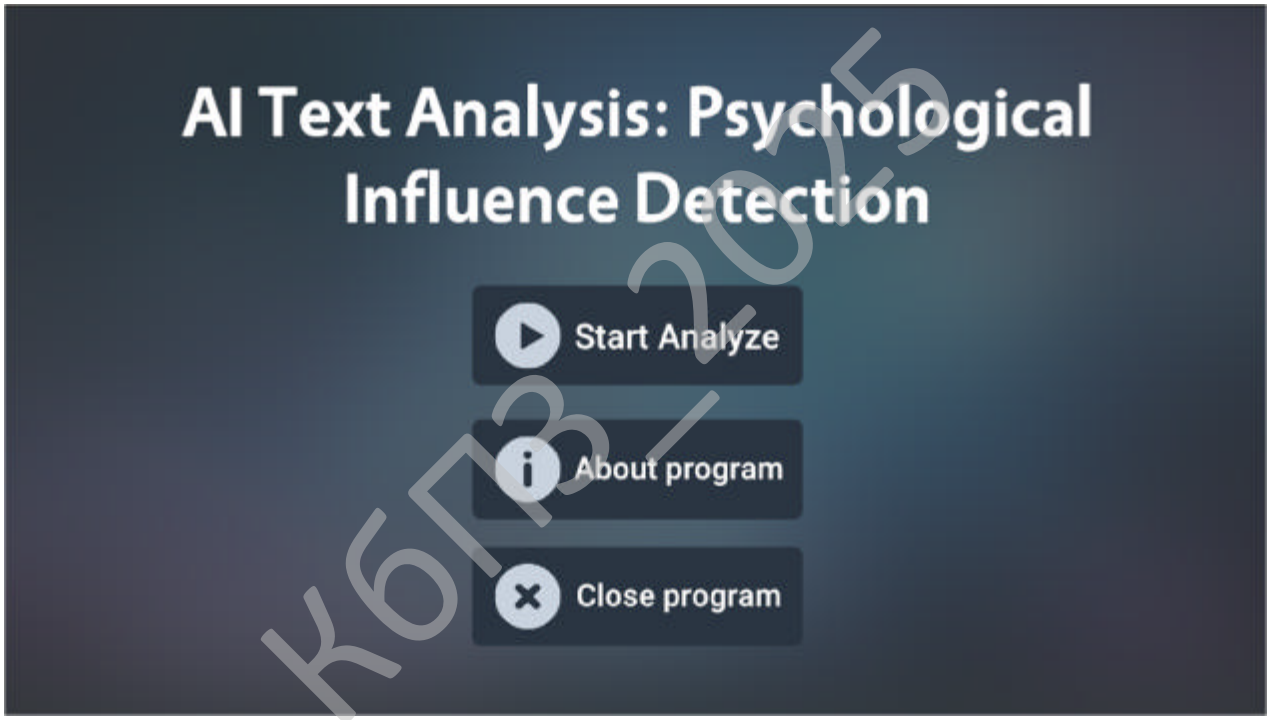




Рисунок 5.1 – Головне меню програми

Після натискання на кнопку початку аналізу відкриється основне вікно з такими блоками:

– Поле для вводу даних.

– Лічильник слів.

– Кнопка для початку аналізу даних.

– Кнопка для очищення поля вводу.

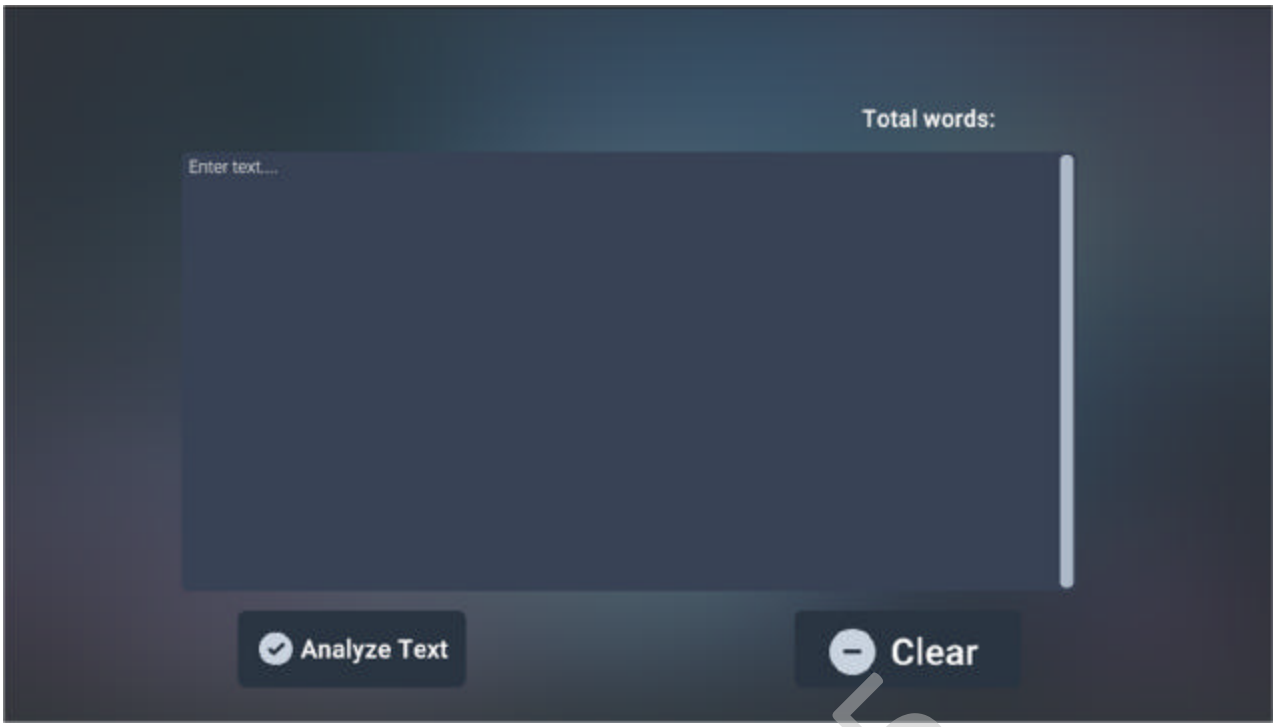


Рисунок 5.2 – Основне вікно програми

На рисунках 5.3 и 5.4 показано результат роботи розробленої програми. Для прикладу було взято уривок з опису сюжету фільму «Angels & Demons».

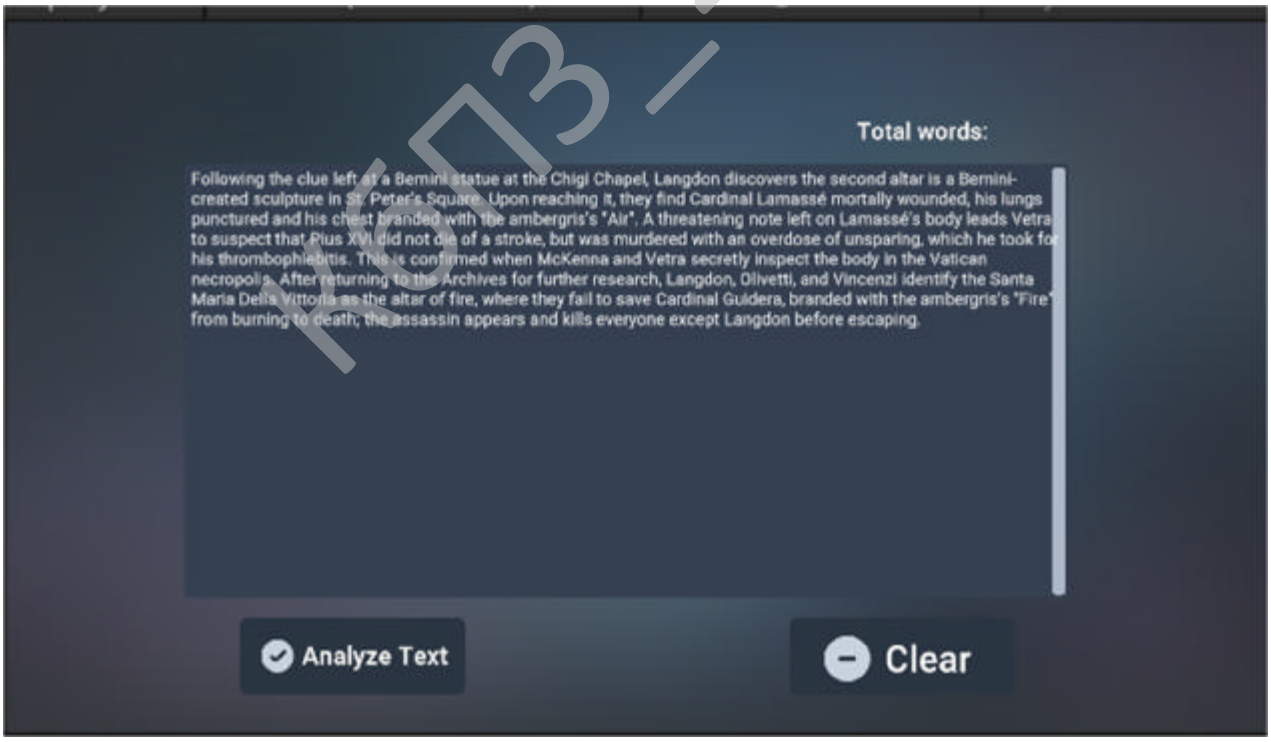


Рисунок 5.3 – Введення тексту в поле

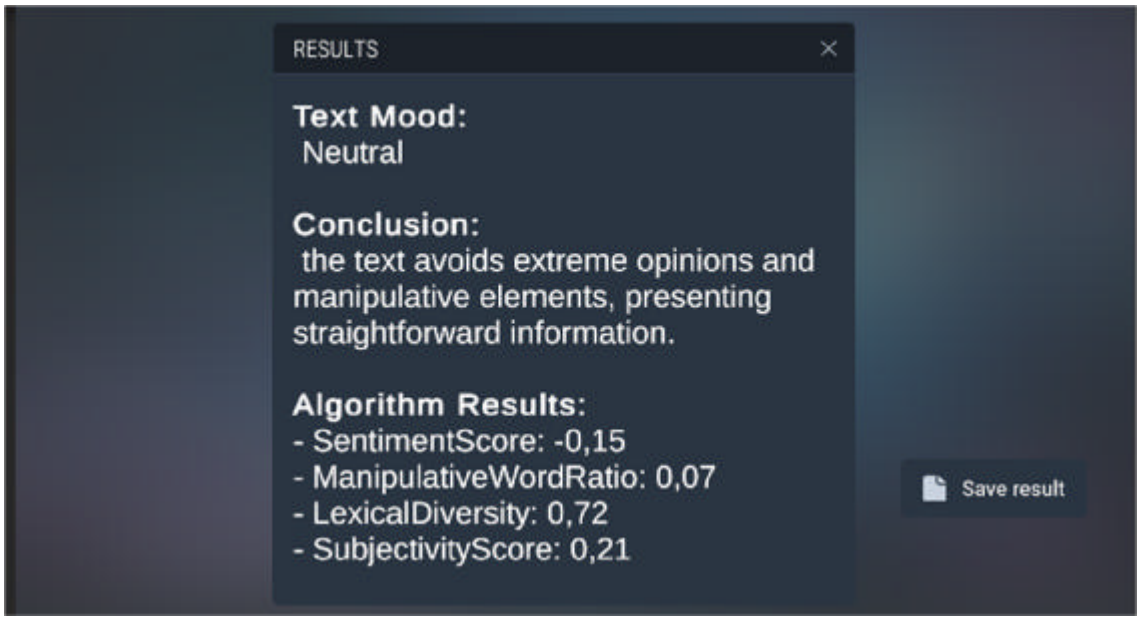


Рисунок 5.3 - Результат аналізу текста

Нейронна мережа класифікувала текст як нейтральний та робить висновок що текст уникає крайніх думок і маніпулятивних елементів, подає відверту інформацію.

Впровадження розробленого програмного забезпечення системи штучного інтелекту для аналізу текстів на наявність інформаційно-психологічних впливів в промислову експлуатацію не є ресурсоємким і не потребує значних ресурсів. Програмне забезпечення розгортається та функціонує на персональному комп’ютері і не потребує додаткових обчислювальних можливостей.

Перед початком використання необхідно буде виконати встановлення та налаштування даного програмного забезпечення на персональних комп’ютерах, на яких воно планує використовуватись.

Важливим етапом буде проведення тестування системи у виробничому середовищі для виявлення та усунення будь-яких проблем.

Через простий інтуїтивно зрозумілий інтерфейс потреби в окремому інструктажі або навчальних сесій для потенційних користувачів щодо використання даного програмного засобу немає.

# 6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної роботи, призначено для аналізу текстів на наявність інформаційно-психологічних впливів

Для вирішення поставленої мети було проведено:

– дослідження існуючих систем аналізу текстів на наявність інформаційно-психологічних впливів.

– розробку алгоритмів системи аналізу текстів на наявність інформаційно-психологічних впливів.

– реалізацію програмного забезпечення системи штучного інтелекту для аналізу текстів на наявність інформаційно-психологічних впливів.

Реалізовані під час виконання кваліфікаційної роботи алгоритми дозволяють успішно вирішують завдання аналізу текстів на наявність інформаційно-психологічних впливів.

Розроблене програмне забезпечення представляє собою додаток, що можна використовувати практично на будь-якому сучасному комп’ютері.

Програмне забезпечення розроблялося у об’єктно-орієнтованій парадигмі, що робить його гнучким та зручним для підтримки та масштабування.

Для розробки  програмного забезпечення системи штучного інтелекту для аналізу текстів на наявність інформаційно-психологічних впливів. використовувалися мови програмування C# та Python. Дані мови програмування дозволили ефективно вирішити поставлену мету та задачі кваліфікаційної роботи.

У п’ятому розділі пояснювальної записки надаються усі необхідні рекомендації з встановлення серверного програмного забезпечення та інструкція для використання клієнтського програмного забезпечення.

Для захисту розробленого програмного забезпечення було обрано вільну ліцензію Creative Commons.

Програмне забезпечення системи штучного інтелекту для аналізу текстів на наявність інформаційно-психологічних впливів є важливим інструментом у сучасному інформаційному середовищі. Воно може застосовуватися як у медійному аналізі, так і для моніторингу соціальних мереж та виявлення маніпулятивного контенту. З огляду на зростаючий обсяг дезінформації та складність інформаційно-психологічних загроз, використання подібних систем стає не просто рекомендацією, а необхідністю. Тож, розроблене у цій кваліфікаційній роботі програмне забезпечення має важливе практичне значення для забезпечення інформаційної безпеки та протидії маніпулятивним впливам.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Аналіз тональності тексту [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/w/index.php?title=Аналіз_тональності_тексту&oldid=44485908.

2. Sentiment Analysis using TextBlob [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/towards-data-science/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524.

3. Vader: A Comprehensive Guide to Sentiment Analysis in Python [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@rslavanyageetha/vader-a-comprehensive-guide-to-sentiment-analysis-in-python-c4f1868b0d2e.

4. How to build: VADER sentiment analysis [Електронний ресурс]. – Режим доступу: https://hex.tech/templates/sentiment-analysis/vader-sentiment-analysis/#:~:text=In%20identifying%20the%20sentiment%20of,sentiment%20score%20of%20the%20text.

5. Python [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/w/index.php?title=Python&oldid=44658602.

6. Hopf M. NLP With Google Cloud Natural Language API [Електронний ресурс] / Maximilian Hopf. – Режим доступу: https://www.toptal.com/machine-learning/google-nlp-tutorial.

7. Flair [Електронний ресурс]. – Режим доступу: https://github.com/flairNLP/flair?tab=readme-ov-file.

8. Type Token Ratio in NLP [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@rajeswaridepala/empirical-laws-ttr-cc9f826d304d.

9. SpaCy [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/w/index.php?title=SpaCy&oldid=41743063.

10. What is Amazon Comprehend? [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/comprehend/latest/dg/what-is.html.

11. Adieu, Text Analysis API [Електронний ресурс]. – Режим доступу: https://aylien.com/blog/adieu-text-analysis-api.

12. Perspective API [Електронний ресурс]. – Режим доступу: https://www.perspectiveapi.com/how-it-works/.

13. Semantic text analysis istio [Електронний ресурс]. – Режим доступу: https://istio.com/text/analyz

14. Natural Language Toolkit [Електронний ресурс]. – Режим доступу: https://www.nltk.org/.

15. Guide | TensorFlow Core [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/guide

16. TextRazor API [Електронний ресурс]. – Режим доступу: https://www.textrazor.com/.

17. TextBlob: Simplified Text Processing [Електронний ресурс]. – Режим доступу: https://textblob.readthedocs.io/en/dev/.

18. TensorFlow [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/w/index.php?title=TensorFlow&oldid=43191359.

19. Introduction to TensorFlow - GeeksforGeeks [Електронний ресурс] // GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/introduction-to-tensorflow/.

20. Angels & Demons [Електронний ресурс]. – Режим доступу: https://www.imdb.com/title/tt0808151/.

21. Unity Documentation [Електронний ресурс] // Unity Documentation. – Режим доступу: https://docs.unity.com/.

22. 3.13.2 Documentation [Електронний ресурс] // 3.13.2 Documentation. – Режим доступу: https://docs.python.org/3/.

23. Microsoft Visual Studio – Вікіпедія [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio.

24. Microsoft Visual Studio: що це, для чого це потрібно та як це працює [Електронний ресурс]. – Режим доступу: https://macrosoft.store/uk/blog/post/29-для-чого-потрібна-microsoft-visual-studio.

25. Unity (ігровий рушій) – Вікіпедія [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Unity_(ігровий_рушій).

26. DeText: A deep NLP framework for intelligent text understanding [Електронний ресурс]. – Режим доступу: https://www.linkedin.com/blog/engineering/open-source/open-sourcing-detext.

27. GitHub - linkedin/detext: DeText: A Deep Neural Text Understanding Framework for Ranking and Classification Tasks [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/linkedin/detext.

28. Кобільник Б. Роль інформаційно-психологічних впливів у інформаційній війні [Електронний ресурс] / Б. Кобільник, А. Гізун // Актуальні задачі та досягнення у галузі кібербезпеки : Матеріали Всеукр. науково-практ. конф., Кропивницький, 23 листоп. 2016 р. – Кропивницький, 2016. – С. 28–29. – Режим доступу: https://dspace.kntu.kr.ua/handle/123456789/5130.

29. Штучний інтелект і дезінформація: можливості та ризики в умовах війни [Електронний ресурс] // Центр стратегічних комунікацій. – Режим доступу: https://spravdi.gov.ua/shtuchnyj-intelekt-i-dezinformacziya-mozhlyvosti-ta-ryzyky-v-umovah-vijny/.

30. Інтелектуальний аналіз тексту – Вікіпедія [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Інтелектуальний_аналіз_тексту.

31. Тема 5. Проектування програмного забезпечення при структурному підході [Електронний ресурс] // StudFiles. – Режим доступу: https://studfile.net/preview/5484923/.

32. CC BY-SA 4.0 Attribution-ShareAlike 4.0 International [Електронний ресурс] // Creative Commons. – Режим доступу: https://creativecommons.org/licenses/by-sa/4.0/.

33. Detecting subjectivity and tone with automated text analysis tools [Електронний ресурс] // Pew Research Center. – Режим доступу: https://www.pewresearch.org/decoded/2018/07/26/detecting-subjectivity-and-tone-with-automated-text-analysis-tools/.

34. Молодецька-Гринчук К. В. Метод виявлення ознак інформаційних впливів у соціальних інтернет-сервісах за змістовними ознаками / К. В. Молодецька-Гринчук // Радіоелектроніка, інформатика, управління. – 2017. – № 2 (41). – С. 117–126.

35. Perkhach R.-Y. LINGUISTIC METHODS OF MANIPULATION IN PRESS: CONTENT ANALYSIS [Електронний ресурс] / Roksolana-Yustyna Perkhach, Alona Smyrnova // Young Scientist. – 2019. – Т. 10, № 74. – Режим доступу: https://doi.org/10.32839/2304-5809/2019-10-74-43

36. Holota T. Manipulative messages in the headlines of media sources [Електронний ресурс] / Taras Holota // Ukrainian Linguistics. – 2023. – № 53. – С. 135–148. – Режим доступу: https://doi.org/10.17721/um/53(2023).135-148.

37. Tereshchenko L. Неправдивість письмового тексту: шляхи її визначення [Електронний ресурс] / Liliia Tereshchenko, Svitlana Gladio // PSYCHOLINGUISTICS. – 2022. – Т. 31, № 2. – С. 116–136. – Режим доступу: https://doi.org/10.31470/2309-1797-2022-31-2-116-136.

38. Відкрите програмне забезпечення – Вікіпедія [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Відкрите_програмне_забезпечення.

39. Ліцензії Creative Commons – Вікіпедія [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Ліцензії_Creative_Commons.

40. Що таке Creative Commons [Електронний ресурс] – Режим доступу до ресурсу: https://www.creativecommons.org.ua/about-creative-commons.

41. Lexical density - Wikipedia [Електронний ресурс] // Wikipedia, the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Lexical_density.

42. Маніпуляції в медіа [Електронний ресурс] // Blank page. – Режим доступу: https://j-line.tilda.ws/media-manipulation.

43. Type/token ratio (TTR) | Sketch Engine [Електронний ресурс] // Sketch Engine. – Режим доступу: https://www.sketchengine.eu/glossary/type-token-ratio-ttr/.

44. Cruickshank A. Analytics is Subjective [Електронний ресурс] / Alan Cruickshank // Desiging Overload. – Режим доступу: https://designingoverload.com/analytics-is-subjective/.

45. Muhammad U. S. How to Perform Sentiment Analysis using Python: Step-by-Step Tutorial with Code Snippets [Електронний ресурс] / Umar Sani Muhammad // Medium. – Режим доступу: https://medium.com/@umarsmuhammed/how-to-perform-sentiment-analysis-using-python-step-by-step-tutorial-with-code-snippets-4ac3e9747fff.

46. Sentiment Analysis Using Python [Електронний ресурс] // Analytics Vidhya. – Режим доступу: https://www.analyticsvidhya.com/blog/2022/07/sentiment-analysis-using-python/.

47. Maxym Z. Діаграми потоків даних (Data Flow Diagrams) [Електронний ресурс] / Zosym Maxym // Maxym Zosym. – Режим доступу: https://www.maxzosim.com/data-flow-diagrams/.

48. Блок-схема: основні елементи та застосування в розробці ПО [Електронний ресурс] // FoxmindEd. – Режим доступу: https://foxminded.ua/blok-skhema/.

49. Ulichev O. Program modeling dissemination of information-psychological influences in virtual social networks [Електронний ресурс] / Oleksandr Ulichev,

Yelyzaveta Meleshko // Advanced Information Systems. – 2018. – Т. 2, № 2. – С. 35–39. – Режим доступу: https://doi.org/10.20998/2522-9052.2018.2.06.

50. Voitovych O. Negative Influences Exposure in Social Internet-services [Електронний ресурс] / Olesia Voitovych, Veronika Ostrovska, Igor Zakalov // Digital Platform: Information Technologies in Sociocultural Sphere. – 2018. – № 2. – С. 93–105. – Режим доступу: https://doi.org/10.31866/2617-796x.2.2018.155667.

51. Pokrovska V. Analysis of information-psychological impact detection methods in social networks [Електронний ресурс] / Valeriia Pokrovska // Collection "Information Technology and Security". – 2020. – Т. 8, № 1. – С. 40–48. – Режим доступу: https://doi.org/10.20535/2411-1031.2020.8.1.218002.

Додаток А
(обов’язковий)

**Технічне завдання**

**Зміст**

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку нейронної мережі для аналізу текстів на наявність інформаційно-психологічних впливів.

## 2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №48-02 від 17.01.2025 року, видане на кафедрі кібербезпеки та програмного забезпечення.

## 3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж.

## 4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

## 5 Технічні вимоги

### 5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

### 5.2 Показники призначення

Система повинна забезпечувати:

– нейронну мережу для аналізу текстів на наявність інформаційно-психологічних впливів;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

### 5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

### 5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

### 5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

### 5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

### 5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

### 5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

#### 5.8.1 Обладнання

Комп’ютер Intel Core i5-10210U/ RAM 8 ГБ / SSD 512 ГБ / NVIDIA GeForce MX250, 2 ГБ або сумісні з ним.

#### 5.8.2 Мова програмування

Програму розроблено на мовах програмування C# та Pyhon.

### 5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

### 5.8.4 Вихідні дані

Робоча програма.

## 6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

## 7 Перелік документів, що розробляються

– Структурна схема системи.
– Функціональна схема системи.
– Діаграма процесів.
– Блок-схема алгоритму роботи програми.
– Пояснювальна записка.

## 8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 24.05.2025 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист __.__.2025 р.

Додаток Б

(обов’язковий)

**Міністерство освіти і науки України**

**Центральноукраїнський національний технічний університет**

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ Є.В. Мелешко

***Програмне забезпечення для аналізу текстів на наявність інформаційно-психологічних впливів за допомогою нейронних мереж***

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 21

Літера: РП



Кропивницький – 2025 року

**// ai_model.py – Скрипт нейроної мережі**

```
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import json
import os

# File paths (Unity adaptation)
current_path = os.path.abspath(__file__)
while not current_path.endswith("Assets"):
    current_path = os.path.dirname(current_path)

unity_assets_path = os.path.join(current_path, "AIModels")
model_path = os.path.join(unity_assets_path, "text_model.keras")
dataset_path = os.path.join(current_path, "Dataset", "Dataset.json")

# Load dataset
if not os.path.exists(dataset_path):
    raise FileNotFoundError(f"Dataset not found at: {dataset_path}")

with open(dataset_path, "r", encoding="utf-8") as file:
    data = json.load(file)

# Load input data
X = np.array([[d["SentimentScore"], d["ManipulativeWordRatio"],
d["LexicalDiversity"], d["SubjectivityScore"]] for d in data])
y_labels = [d["Label"] for d in data]
y_conclusions = [d["Conclusion"] for d in data]

# Normalize data
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

# Encode labels
label_encoder = LabelEncoder()
y_labels_encoded = label_encoder.fit_transform(y_labels)  # Now integers

conclusion_encoder = LabelEncoder()
y_conclusions_encoded = conclusion_encoder.fit_transform(y_conclusions)  #
Now integers

# Create or load model
if os.path.exists(model_path):
    model = tf.keras.models.load_model(model_path)
else:
    input_layer = tf.keras.layers.Input(shape=(4,))
    hidden_layer = tf.keras.layers.Dense(8, activation="relu")(input_layer)

    label_output = tf.keras.layers.Dense(len(set(y_labels)),
activation="softmax", name="label_output")(hidden_layer)
    conclusion_output = tf.keras.layers.Dense(len(set(y_conclusions)),
activation="softmax", name="conclusion_output")(hidden_layer)

    model = tf.keras.Model(inputs=input_layer, outputs=[label_output,
conclusion_output])
    model.compile(optimizer="adam",
                  loss=["sparse_categorical_crossentropy",
"sparse_categorical_crossentropy"],
                  metrics=["accuracy", "accuracy"])

    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor="loss", patience=20, restore_best_weights=True
    )

    model.fit(X, [y_labels_encoded, y_conclusions_encoded], epochs=150,
```

```
batch_size=8, callbacks=[early_stopping])
    model.save(model_path)

# Prediction function
def predict_text(json_file_path):
    try:
        with open(json_file_path, "r", encoding="utf-8-sig") as file:
            data = json.load(file)

        input_data = np.array([[data[key] for key in ["SentimentScore",
"ManipulativeWordRatio", "LexicalDiversity", "SubjectivityScore"]]])
        input_data = scaler.transform(input_data)

        pred_label, pred_conclusion = model.predict(input_data)

        label_index = np.argmax(pred_label)
        conclusion_index = np.argmax(pred_conclusion)

        decoded_label = label_encoder.inverse_transform([label_index])[0]
        decoded_conclusion =
conclusion_encoder.inverse_transform([conclusion_index])[0]

        print(f"[INFO] Prediction: Label={decoded_label},
Conclusion={decoded_conclusion}")
        return str(decoded_label), str(decoded_conclusion)

    except Exception as e:
        print(f"[ERROR] Python Error: {e}")
        return "Python Error", str(e)
```

**// TextClassifier.cs – Скрипт призначений для інтеграції нейромережевої моделі, написаної на Python, у Unity.**

```
using System;
using System.IO;
using Newtonsoft.Json;
using Python.Runtime;
using UnityEngine;

namespace AIScripts
{
    public class TextClassifier
    {
        private PyObject predictFunction;

        public TextClassifier()
        {
            try
            {
                string pythonCodePath =
Path.Combine(Application.streamingAssetsPath, "PythonCode");
                string pythonHome =
Path.Combine(Application.streamingAssetsPath, "Python");
                string pythonLib = Path.Combine(pythonHome, "Lib");
                string pythonSitePackages = Path.Combine(pythonLib, "site-
packages");

                Environment.SetEnvironmentVariable("PYTHONHOME", pythonHome,
EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONPATH", pythonLib +
";" + pythonSitePackages, EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONNET_PYDLL",
Path.Combine(pythonHome, "python311.dll"),
EnvironmentVariableTarget.Process);

                PythonEngine.Initialize();
                Debug.Log("Python Engine initialized.");

                using (Py.GIL())
                {
                    PyObject sys = Py.Import("sys");
                    PyObject sysPath = sys.GetAttr("path");
                    sysPath.InvokeMethod("append", new
PyString(pythonCodePath));

                    PyObject aiModule = Py.Import("ai_model");
                    predictFunction = aiModule.GetAttr("predict_text");

                    Debug.Log("Python function predict_text() loaded
successfully.");
                }
            }
            catch (Exception ex)
            {
                Debug.LogError($"Error initializing Python: {ex.Message}");
            }
        }

        public (string, string) PredictText(double sentiment, double
manipulative, double lexical, double subjectivity)
        {
            using (Py.GIL())
            {
                try
                {
                    if (predictFunction == null)
```

```
                    {
                        Debug.LogError("predictFunction is NULL. Python
module might not be loaded.");
                        return ("Error", "Failed to process");
                    }

                    string jsonFilePath =
Path.Combine(Application.streamingAssetsPath, "AnalysisResults.json");

                    var inputData = new
                    {
                        SentimentScore = sentiment,
                        ManipulativeWordRatio = manipulative,
                        LexicalDiversity = lexical,
                        SubjectivityScore = subjectivity
                    };

                    string jsonData = JsonConvert.SerializeObject(inputData,
Formatting.Indented);

                    File.WriteAllText(jsonFilePath, jsonData,
Encoding.UTF8);
                    Debug.Log($"JSON saved at: {jsonFilePath}");
                    Debug.Log($"JSON content: {jsonData}");

                    PyObject result = predictFunction.Invoke(new
PyString(jsonFilePath));

                    if (result == null)
                    {
                        Debug.LogError("Python function returned NULL.");
                        return ("Error", "Python function failed");
                    }

                    string label = result[0].As<string>();
                    string conclusion = result[1].As<string>();

                    Debug.Log($"Label: {label}");
                    Debug.Log($"Conclusion: {conclusion}");

                    return (label, conclusion);
                }
                catch (Exception ex)
                {
                    Debug.LogError($"Error when calling Python function:
{ex.Message}");
                    return ("Error", "Failed to process");
                }
            }
        }
    }
}
```

**// SentimentAnalysis.cs – Скрипт для аналізу тональності тексту**

```
using System;
using System.IO;
using System.Text.RegularExpressions;
using Python.Runtime;
using UnityEngine;

namespace AnalysisScripts
{
    public class SentimentAnalyzer
    {
        private PyObject textBlob;

        public SentimentAnalyzer()
        {
            try
            {
                string pythonHome =
Path.Combine(Application.streamingAssetsPath, "Python");
                string pythonLib = Path.Combine(pythonHome, "Lib");
                string pythonSitePackages = Path.Combine(pythonLib, "site-
packages");

                Environment.SetEnvironmentVariable("PYTHONHOME", pythonHome,
EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONPATH", pythonLib +
";" + pythonSitePackages, EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONNET_PYDLL",
Path.Combine(pythonHome, "python311.dll"),
EnvironmentVariableTarget.Process);

                PythonEngine.Initialize();
                Debug.Log("Python Engine initialized.");

                using (Py.GIL())
                {
                    PyObject sys = Py.Import("sys");
                    PyObject sysPath = sys.GetAttr("path");
                    sysPath.InvokeMethod("append", new
PyString(pythonSitePackages));
                    Debug.Log($"Added path: {pythonSitePackages}");

                    textBlob = Py.Import("textblob");
                }
            }
            catch (Exception ex)
            {
                Debug.LogError($"Initialization error Python:
{ex.Message}");
            }
        }

        public (double FinalSentiment, double VaderScore, double
TextBlobScore) CalculateSentimentScore(string text, double
manipulativeRatio)
        {
            text = Regex.Replace(text, @"[^\w\s.!?]", "").ToLower().Trim();

            if (string.IsNullOrWhiteSpace(text))
            {
                Debug.LogError("Input text is empty!");
                return (0.0, 0.0, 0.0);
            }

            double totalVader = 0.0;
```

```
            double totalTextBlob = 0.0;
            int sentenceCount = 0;

            string[] sentences = text.Split(new char[] { '.', '!', '?' },
StringSplitOptions.RemoveEmptyEntries);

            using (Py.GIL())
            {
                PyObject vader = Py.Import("vaderSentiment.vaderSentiment");
                PyObject SentimentIntensityAnalyzer =
vader.GetAttr("SentimentIntensityAnalyzer");
                PyObject analyzer = SentimentIntensityAnalyzer.Invoke();
                PyObject TextBlob = textBlob.GetAttr("TextBlob");

                foreach (string sentence in sentences)
                {
                    if (sentence.Trim().Length > 1)
                    {
                        PyObject scores =
analyzer.InvokeMethod("polarity_scores", new PyString(sentence));
                        double compound =
scores.GetItem("compound").As<double>();
                        totalVader += compound;

                        PyObject blob = TextBlob.Invoke(new
PyString(sentence));
                        PyObject sentiment = blob.GetAttr("sentiment");
                        double textBlobPolarity =
sentiment.GetAttr("polarity").As<double>();

                        textBlobPolarity = double.IsNaN(textBlobPolarity) ?
0.0 : textBlobPolarity;

                        totalTextBlob += textBlobPolarity;
                        sentenceCount++;
                    }
                }
            }

            double vaderScore = sentenceCount > 0 ? totalVader /
sentenceCount : 0.0;
            double textBlobScore = sentenceCount > 0 ? totalTextBlob /
sentenceCount : 0.0;

            vaderScore = Math.Max(-1, Math.Min(vaderScore, 1));
            textBlobScore = Math.Max(-1, Math.Min(textBlobScore, 1));

            double finalSentiment = (vaderScore + textBlobScore) / 2;

            double penalty = manipulativeRatio > 0.5 ? (manipulativeRatio >
0.9 ? 0.15 : manipulativeRatio > 0.7 ? 0.1 : 0.05) : 0;
            finalSentiment -= penalty;

            finalSentiment = Math.Max(-1, Math.Min(finalSentiment, 1));

            Debug.Log($"Final Sentiment: {finalSentiment}, Vader:
{vaderScore}, TextBlob: {textBlobScore}, ManipulativePenalty: -{penalty}");

            return (Math.Round(finalSentiment, 2), Math.Round(vaderScore,
2), Math.Round(textBlobScore, 2));
        }
    }
}
```

**// ManipulativeWordAnalysis.cs – Скрипт для аналізу тексту на наявність маніпулятивних слів та фраз**

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using UnityEngine;
using Newtonsoft.Json;

namespace AnalysisScripts
{
    public class ManipulativeWordAnalysis : MonoBehaviour
    {
        private static HashSet<string> manipulativeWords = new
HashSet<string>();
        private static List<string> manipulativePhrases = new
List<string>();

        private string filePath;

        async void Start()
        {
            filePath = Path.Combine(Application.dataPath,
"Resources/manipulative_words.json");
            await LoadManipulativeWordsAsync();
        }

        private async Task LoadManipulativeWordsAsync()
        {
            if (!File.Exists(filePath))
            {
                return;
            }

            try
            {
                string jsonContent = await File.ReadAllTextAsync(filePath);
                var wordData =
JsonConvert.DeserializeObject<ManipulativeWordsWrapper>(jsonContent);

                manipulativeWords.Clear();
                manipulativePhrases.Clear();

                AddWordsFromCategory(wordData.generalization);
                AddWordsFromCategory(wordData.call_to_action);
                AddWordsFromCategory(wordData.emotional_triggers);
                AddWordsFromCategory(wordData.fear_based);
                AddWordsFromCategory(wordData.manipulative_contrasts);
                AddWordsFromCategory(wordData.false_analogies);
                AddWordsFromCategory(wordData.concept_substitution);
                AddWordsFromCategory(wordData.appeal_to_patriotism);
                AddWordsFromCategory(wordData.doubt_and_uncertainty);
                AddWordsFromCategory(wordData.urgency_and_alarmism);

                Debug.Log($"Loaded: {manipulativeWords.Count} words,
{manipulativePhrases.Count} phrases.");
            }
            catch (Exception ex)
            {
                Debug.LogError($"Error loading manipulative words:
{ex.Message}");
            }
        }
```

```
        private void AddWordsFromCategory(List<string> words)
        {
            if (words == null) return;

            foreach (var phrase in words)
            {
                if (phrase.Contains(" "))
                    manipulativePhrases.Add(phrase.ToLower());
                else
                    manipulativeWords.Add(phrase.ToLower());
            }
        }

        public static double CalculateManipulativeWordRatio(string text)
        {
            if (manipulativeWords.Count == 0 && manipulativePhrases.Count ==
0)
            {
                return 0;
            }

            string cleanedText = Regex.Replace(text, @"[^\w\s-]", "");
            string[] words = cleanedText.Split(new char[] { ' ', '\n', '\r',
'\t' }, StringSplitOptions.RemoveEmptyEntries);

            int manipulativeCount = 0;
            HashSet<string> countedWords = new HashSet<string>();
            HashSet<string> countedPhrases = new HashSet<string>();

            foreach (var word in words)
            {
                if (word.Length > 3 && manipulativeWords.Contains(word) &&
!countedWords.Contains(word))
                {
                    manipulativeCount++;
                    countedWords.Add(word);
                }
            }

            foreach (var phrase in manipulativePhrases)
            {
                string[] phraseWords = phrase.Split(' ');

                int matchCount = phraseWords.Count(w => words.Contains(w));
                double matchRatio = (double)matchCount / phraseWords.Length;

                if (matchRatio >= 0.65 && !countedPhrases.Contains(phrase))
                {
                    manipulativeCount++;
                    countedPhrases.Add(phrase);
                }
            }

            Debug.Log($"Total words: {words.Length}, Unique manipulative
words/phrases found: {manipulativeCount}");

            return words.Length > 0 ? Math.Round((double)manipulativeCount /
words.Length, 2) : 0;
        }
    }

    [Serializable]
    public class ManipulativeWordsWrapper
    {
        public List<string> generalization;
```

```
        public List<string> call_to_action;
        public List<string> emotional_triggers;
        public List<string> fear_based;
        public List<string> manipulative_contrasts;
        public List<string> false_analogies;
        public List<string> concept_substitution;
        public List<string> appeal_to_patriotism;
        public List<string> doubt_and_uncertainty;
        public List<string> urgency_and_alarmism;
    }
}
```

**// LexicalDiversityAnalysis.cs – Скрипт для аналізу лексичного різноманіття тексту**

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using UnityEngine;

namespace AnalysisScripts
{
    public class LexicalDiversityAnalysis : MonoBehaviour
    {
        public static double CalculateLexicalDiversity(string text)
        {
            text = Regex.Replace(text, @"[^\w\s]", "").ToLower();

            string[] words = text.Split(new char[] { ' ', '\n', '\r', '\t'
}, StringSplitOptions.RemoveEmptyEntries);
            HashSet<string> uniqueWords = new HashSet<string>(words);

            if (words.Length > 0)
            {
                return Math.Round((double)uniqueWords.Count / words.Length,
2);
            }
            else
            {
                return 0;
            }
        }
    }
}
```

**// SubjectivityAnalyzer.cs – Скрипт для аналізу суб'єктивності тексту**

```
using System;
using System.IO;
using System.Text.RegularExpressions;
using Python.Runtime;
using UnityEngine;

namespace AnalysisScripts
{
    public class SubjectivityAnalyzer
    {
        private PyObject textBlob;

        public SubjectivityAnalyzer()
        {
            try
            {
                string pythonHome =
Path.Combine(Application.streamingAssetsPath, "Python");
                string pythonLib = Path.Combine(pythonHome, "Lib");
                string pythonSitePackages = Path.Combine(pythonLib, "site-
packages");

                Environment.SetEnvironmentVariable("PYTHONHOME", pythonHome,
EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONPATH", pythonLib +
";" + pythonSitePackages, EnvironmentVariableTarget.Process);
                Environment.SetEnvironmentVariable("PYTHONNET_PYDLL",
Path.Combine(pythonHome, "python311.dll"),
EnvironmentVariableTarget.Process);

                PythonEngine.Initialize();
                Debug.Log("Python Engine initialized.");

                using (Py.GIL())
                {
                    PyObject sys = Py.Import("sys");
                    PyObject sysPath = sys.GetAttr("path");
                    sysPath.InvokeMethod("append", new
PyString(pythonSitePackages));
                     Debug.Log($"Added path: {pythonSitePackages}");

                    textBlob = Py.Import("textblob");
                }
            }
            catch (Exception ex)
            {
                Debug.LogError($"Initialization error Python:
{ex.Message}");
            }
        }

        public double CalculateSubjectivity(string text)
        {
            text = Regex.Replace(text, @"[^\w\s.!?]", "").ToLower().Trim();

            if (string.IsNullOrWhiteSpace(text))
            {
                Debug.LogError("Input text is empty!");
                return 0.0;
            }

            double totalSubjectivity = 0.0;
            int sentenceCount = 0;
```

```
            string[] sentences = text.Split(new char[] { '.', '!', '?' },
StringSplitOptions.RemoveEmptyEntries);

            using (Py.GIL())
            {
                PyObject TextBlob = textBlob.GetAttr("TextBlob");

                foreach (string sentence in sentences)
                {
                    if (sentence.Trim().Length > 1)
                    {
                        PyObject blob = TextBlob.Invoke(new
PyString(sentence));
                        PyObject sentiment = blob.GetAttr("sentiment");
                        double subjectivity =
sentiment.GetAttr("subjectivity").As<double>();

                        subjectivity = double.IsNaN(subjectivity) ? 0.0 :
subjectivity;

                        totalSubjectivity += subjectivity;
                        sentenceCount++;
                    }
                }
            }

            double finalSubjectivity = sentenceCount > 0 ? totalSubjectivity
/ sentenceCount : 0.0;
            finalSubjectivity = Math.Max(0, Math.Min(finalSubjectivity, 1));

            Debug.Log($"Subjectivity Score: {finalSubjectivity}");

            return Math.Round(finalSubjectivity, 2);
        }
    }

}
```

**// ButtonAnimation.cs – Скрипт керує анімацією та обробляє натискання кнопок у головному меню**

```
using UnityEngine;
using UnityEngine.SceneManagement;
using Ricimi;
using DG.Tweening;

namespace UI
{
    public class ButtonAnimation : MonoBehaviour
    {
        private bool isClicked = false;
        private CleanButton button;

        public CanvasGroup mainMenuCanvas;
        public CanvasGroup aboutCanvas;
        public GameObject closeButton;

        private void Start()
        {
            button = GetComponent<CleanButton>();

            aboutCanvas.alpha = 0;
        }

        public void OnButtonClick()
        {
            if (isClicked) return;
            isClicked = true;

            string buttonName = gameObject.name;
            button.interactable = false;

            transform.DOScale(0.9f, 0.1f).OnComplete(() =>
            {
                transform.DOScale(1f, 0.1f).OnComplete(() =>
                {
                    if (buttonName == "StartButton")
                    {
                        SceneManager.LoadScene("Program");
                    }
                    else if (buttonName == "ExitButton")
                    {
                        Application.Quit();
                    }
                    else if (buttonName == "AboutButton")
                    {
                        ShowCanvas(aboutCanvas, closeButton);
                    }
                    else if (buttonName == "CloseButton")
                    {
                        HideCanvas(aboutCanvas, closeButton);
                    }

                    isClicked = false;
                    button.interactable = true;
                });
            });
        }

        private void ShowCanvas(CanvasGroup canvas, GameObject button)
        {
            if (mainMenuCanvas != null)
            {
                mainMenuCanvas.DOFade(0, 0.2f).OnComplete(() =>
```

```
                {
                    mainMenuCanvas.interactable = false;
                    mainMenuCanvas.blocksRaycasts = false;
                    mainMenuCanvas.gameObject.SetActive(false);
                });
            }

            if (canvas != null)
            {
                canvas.gameObject.SetActive(true);
                canvas.DOFade(1f, 0.3f).OnComplete(() =>
                {
                    canvas.interactable = true;
                    canvas.blocksRaycasts = true;
                });
            }

            if (button != null) button.SetActive(true);
        }

        private void HideCanvas(CanvasGroup canvas, GameObject button)
        {
            if (canvas != null)
            {
                canvas.DOFade(0f, 0.2f).OnComplete(() =>
                {
                    canvas.gameObject.SetActive(false);
                    canvas.interactable = false;
                    canvas.blocksRaycasts = false;

                    if (mainMenuCanvas != null)
                    {
                        mainMenuCanvas.gameObject.SetActive(true);
                        mainMenuCanvas.DOFade(1f, 0.3f).OnComplete(() =>
                        {
                            mainMenuCanvas.interactable = true;
                            mainMenuCanvas.blocksRaycasts = true;
                        });
                    }
                });
            }

            if (button != null) button.SetActive(false);
        }
    }
}
```

**// ProgramUI.cs – Скрипт відповідає за управління інтерфейсом користувача в сцені аналізу тексту**

```
using DG.Tweening;
using Managers;
using System.Collections;
using System.IO;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace UI
{
    public class ProgramUI : MonoBehaviour
    {
        public GameObject UIProgramCanvas;
        public GameObject LoadingCanvas;
        public GameObject ResultsCanvas;

        public TextMeshProUGUI resultText;

        public TextAnalysisManager textAnalysisManager;

        private string label = null;
        private string conclusion = null;
        private float sentiment = 0;
        private float manipulative = 0;
        private float lexical = 0;
        private float subjectivity = 0;

        public void StartAnalysis()
        {
            UIProgramCanvas.SetActive(false);
            LoadingCanvas.SetActive(true);
            LoadingCanvas.GetComponent<CanvasGroup>().DOFade(1, 0.5f);

            StartCoroutine(WaitForAnalysisResults());
        }

        public void BackToMenu()
        {
            SceneManager.LoadScene("MainMenu");
        }


        public void SaveResults()
        {
            string savePath = Path.Combine(Application.dataPath,
"SavedResults");

            if (!Directory.Exists(savePath))
            {
                Directory.CreateDirectory(savePath);
            }

            string fileName = $"AnalysisResult_{System.DateTime.Now:yyyy-MM-
dd_HH-mm-ss}.txt";
            string filePath = Path.Combine(savePath, fileName);

            string resultString = $"Text Mood: {label}\n\n" +
                                  $"Conclusion: {conclusion}\n\n" +
                                  $"Algorithm Results:\n" +
                                  $"- SentimentScore: {sentiment:F2}\n" +
                                  $"- ManipulativeWordRatio:
{manipulative:F2}\n" +
                                  $"- LexicalDiversity: {lexical:F2}\n" +
```

```
                                    $"- SubjectivityScore: {subjectivity:F2}";

            File.WriteAllText(filePath, resultString);

            Debug.Log($"Results saved to {filePath}");
        }

        IEnumerator WaitForAnalysisResults()
        {
            textAnalysisManager.AnalyzeText();

            while (!IsDataValid())
            {
                label = textAnalysisManager.GetLabel();
                conclusion = textAnalysisManager.GetConclusion();
                sentiment = textAnalysisManager.GetSentimentScore();
                manipulative = textAnalysisManager.GetManipulativeRatio();
                lexical = textAnalysisManager.GetLexicalDiversity();
                subjectivity = textAnalysisManager.GetSubjectivityScore();

                yield return null;
            }

            yield return new WaitForSeconds(5);

            SwitchToResults();
        }

        void SwitchToResults()
        {
            LoadingCanvas.GetComponent<CanvasGroup>().DOFade(0,
0.5f).OnComplete(() =>
            {
                LoadingCanvas.SetActive(false);

                ResultsCanvas.SetActive(true);
                ResultsCanvas.GetComponent<CanvasGroup>().DOFade(1, 0.5f);

                string resultString = $"<b>Text Mood:</b>\n {label}\n\n" +
                                      $"<b>Conclusion:</b>\n
{conclusion}\n\n" +
                                      $"<b>Algorithm Results:</b>\n" +
                                      $"- SentimentScore: {sentiment:F2}\n"
+
                                      $"- ManipulativeWordRatio:
{manipulative:F2}\n" +
                                      $"- LexicalDiversity: {lexical:F2}\n"
+
                                      $"- SubjectivityScore:
{subjectivity:F2}";

                resultText.text = resultString;
            });
        }
        bool IsDataValid()
        {
            return !string.IsNullOrEmpty(label) &&
!string.IsNullOrEmpty(conclusion);
        }    }

}
```

**// LoadingManager.cs – Скрипт відповідає за анімацію завантаження**

```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

namespace Managers
{
    public class LoadingManager : MonoBehaviour
    {
        public Image[] whiteCircles;
        public Image[] darkCircles;
        public float animationSpeed = 0.5f;

        private Coroutine loadingCoroutine;

        public void StartLoading()
        {
            gameObject.SetActive(true);
            if (loadingCoroutine == null)
            {
                loadingCoroutine = StartCoroutine(AnimateSpinner());
            }
        }

        public void StopLoading()
        {
            if (loadingCoroutine != null)
            {
                StopCoroutine(loadingCoroutine);
                loadingCoroutine = null;
            }

            ResetCircles();
            gameObject.SetActive(false);
        }

        private IEnumerator AnimateSpinner()
        {
            int index = 0;
            while (true)
            {
                ResetCircles();

                Color whiteColor = whiteCircles[index].color;
                whiteColor.a = 1f;
                whiteCircles[index].color = whiteColor;

                Color darkColor = darkCircles[index].color;
                darkColor.a = 1f;
                darkCircles[index].color = darkColor;

                yield return new WaitForSeconds(animationSpeed);

                index = (index + 1) % whiteCircles.Length;
            }
        }

        private void ResetCircles()
        {
            foreach (Image circle in whiteCircles)
            {
                Color color = circle.color;
                color.a = 0.3f;
                circle.color = color;
            }
```

```
            foreach (Image circle in darkCircles)
            {
                Color color = circle.color;
                color.a = 0.3f;
                circle.color = color;
            }
        }
    }
}
```

**// TextAnalysisManager.cs – Скрипт відповідає за аналіз тексту в реальному часі та передачу цих данних до нейроної мережі**

```
using System;
using UnityEngine;
using AnalysisScripts;
using TMPro;
using System.Text.RegularExpressions;
using AIScripts;
using UI;

namespace Managers
{
    public class TextAnalysisManager : MonoBehaviour
    {
        [SerializeField] private TMP_InputField inputField;
        [SerializeField] private TextMeshProUGUI totalWords;

        private SentimentAnalyzer sentiment;
        private SubjectivityAnalyzer subjectivity;
        private TextClassifier classifier;

        private string label;
        private string conclusion;
        private double sentimentScore;
        private double manipulativeRatio;
        private double lexicalDiversity;
        private double subjectivityScore;

        void Start()
        {
            sentiment = new SentimentAnalyzer();
            subjectivity = new SubjectivityAnalyzer();
            classifier = new TextClassifier();
        }

        public void AnalyzeText()
        {
            Debug.Log("AnalyzeText() called!");

            string text = inputField.text.ToLower();
            int wordCount = CountWords(text);
            totalWords.text = $"Total words: {wordCount}";

            manipulativeRatio =
ManipulativeWordAnalysis.CalculateManipulativeWordRatio(text);
            lexicalDiversity =
LexicalDiversityAnalysis.CalculateLexicalDiversity(text);

            var result = sentiment.CalculateSentimentScore(text,
manipulativeRatio);
            sentimentScore = result.FinalSentiment;
            subjectivityScore = subjectivity.CalculateSubjectivity(text);

            (label, conclusion) = classifier.PredictText(sentimentScore,
manipulativeRatio, lexicalDiversity, subjectivityScore);
        }

        private int CountWords(string text)
        {
            string cleanedText = Regex.Replace(text, @"[^\w\s-]", "");
            string[] words = cleanedText.Split(new char[] { ' ', '\n', '\r',
'\t' }, StringSplitOptions.RemoveEmptyEntries);
            return words.Length;
        }
```

```
        public string GetLabel()
        {
            return label;
        }

        public string GetConclusion()
        {
            return conclusion;
        }

        public float GetSentimentScore()
        {
            return (float)sentimentScore;
        }

        public float GetManipulativeRatio()
        {
            return (float)manipulativeRatio;
        }

        public float GetLexicalDiversity()
        {
            return (float)lexicalDiversity;
        }

        public float GetSubjectivityScore()
        {
            return (float)subjectivityScore;
        }
    }
}
```