**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Західноукраїнський національний університет**
**Факультет комп'ютерних інформаційних технологій**
Кафедра інформаційно-обчислювальних систем і управління

**ШЕЛЮЖАК Ярослав Сергійович**

# Програмний модуль класифікації елементів анотацій наукових статей на основі штучного інтелекту / Software module for elements classification of scientific papers abstracts based on artificial intelligence

спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41
Я. С. Шелюжак

Науковий керівник:
к.т.н., доцент І. В. Турченко

Кваліфікаційну роботу
допущено до захисту:
«___»_____ 20___ р.

Завідувач кафедри
_____ М. П. Комар

**ТЕРНОПІЛЬ – 2024**

**Факультет комп'ютерних інформаційних технологій**
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ М.П. Комар
«____» _____2023 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**
**ШЕЛЮЖАКУ Ярославу Сергійовичу**
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмний модуль класифікації елементів анотацій наукових статей на основі штучного інтелекту / Software module for elements classification of scientific papers abstracts based on artificial intelligence
керівник роботи Турченко Ірина Василівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 12 грудня 2023 р. № 753.
2. Строк подання студентом закінченої кваліфікаційної роботи 15 травня 2024 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:

- проаналізувати способи представлення природної мови для комп'ютерних систем та підходи до обробки природної мови засобами штучного інтелекту;
- провести аналіз існуючих рішень для класифікації елементів анотацій наукових статей на основі штучного інтелекту;
- зробити постановку задачі дослідження;
- представити структуру програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту;
- розробити архітектуру моделі обробки природної мови для виконання класифікації елементів анотацій наукових статей;
- представити алгоритмічне забезпечення програмного модуля;
- провести навчання моделі обробки природної мови та перевірити точність класифікації;
- розробити програмний модуль та протестувати його.

5. Перелік графічного матеріалу в роботі:

- схема алгоритму роботи програмного модуля класифікації елементів анотації наукових статей на основі штучного інтелекту;

– схема алгоритму навчання класифікатора ОПМ на основі архітектури BERT.

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 12 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів кваліфікаційної роботи | Примітка |
|---|---|---|---|
| 1 | Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи | до 01.01. 2024 р. | |
| 2 | Написання 1 розділу кваліфікаційної роботи | до 01.03. 2024 р. | |
| 3 | Написання 2 розділу кваліфікаційної роботи | до 01.04.2024 р. | |
| 4 | Написання 3 розділу кваліфікаційної роботи | до 01.05. 2024 р. | |
| 5 | Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником | до 15.05.2024 р. | |
| 6 | Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів. | до 20.05.2024 р. | |
| 7 | Перевірка кваліфікаційної роботи на оригінальність тексту у системі «Unicheck». | до 10.06.2024 р. | |
| 8 | Оформлення кваліфікаційної роботи та отримання допуску до захисту | до 14.06.2024 р. | |
| 9 | Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії. | до 14.06. 2024 р. | |

Студент_____ Я. С. Шелюжак
( підпис ) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ І.В. Турченко
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль класифікації елементів анотацій наукових статей на основі штучного інтелекту» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 54 сторінки і містить 18 ілюстрацій, 2 додатки та 26 використаних джерел.

Метою кваліфікаційної роботи є створення програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту.

У роботі використовувалися метод аналізу для дослідження існуючих рішень, метод синтезу для поєднання архітектур нейронних мереж для розпізнавання тексту, метод порівняльного аналізу, методи машинного навчання і глибоких нейронних мереж для обробки природної мови.

Внаслідок виконання роботи розроблено програмний модуль класифікації елементів анотацій наукових статей на основі штучного інтелекту.

Результати роботи можуть бути використані науковцями і дослідниками для автоматизованого огляду літератури, а саме: опрацювання анотацій наукових статей, що допоможе їм швидко знаходити релевантні статті, класифікувати за темами, методами, результатами тощо.

Ключові слова: ОБРОБКА ПРИРОДНОЇ МОВИ, НЕЙРОННІ МЕРЕЖІ, БАГАТОКЛАСОВА КЛАСИФІКАЦІЯ, АНОТАЦІЯ НАУКОВОЇ СТАТТІ, АРХІТЕКТУРА BERT, PUBMED 200K RCT, ПРОГРАМНИЙ МОДУЛЬ.

# ANNOTATION

Qualification work on the topic «Software module for elements classification of scientific papers abstracts based on artificial intelligence» for obtaining a bachelor's degree in the specialty 122 «Computer Science» of the educational program «Computer Science» is written on 54 pages and it contains 18 figures, 2 annexes and 26 references.

The purpose of the qualification work is to create a software module for the classification of elements of abstracts of scientific articles based on artificial intelligence.

The work used an analysis method for researching existing solutions, a synthesis method for combining neural network architectures for text recognition, a comparative analysis method, machine learning methods and deep neural networks for natural language processing.

As a result of the work, a software module for the classification of the elements of the annotations of scientific articles was developed based on artificial intelligence.

The results of the work can be used by scientists and researchers for an automated review of the literature, namely: processing of abstracts of scientific articles, which will help them quickly find relevant articles, classify them by topics, methods, results, etc.

Keywords: NATURAL LANGUAGE PROCESSING, NEURAL NETWORKS, MULTI-CLASS CLASSIFICATION, SCIENTIFIC ARTICLE ABSTRACT, BERT ARCHITECTURE, PUBMED 200K RCT, SOFTWARE MODULE.

ЗМІСТ

Вступ ……..……..  7
1 Аналіз предметної області та постановка задачі дослідження ..... 9
1.1 Основні аспекти обробки природної мови засобами штучного інтелекту . 9
1.2 Огляд і аналіз існуючих рішень …….....……... 15
1.3 Постановка задачі дослідження ……....…….……... 17
2 Структура та алгоритмічне забезпечення програмного модуля класифікації елементів анотацій наукових статей ……... 20
2.1 Структура програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту……... 20
2.2 Архітектура моделі ОПМ на основі BERT із множинним вводом даних .. 23
2.3 Алгоритмічне забезпечення програмного модуля……..... 27
3 Програмно-технологічне забезпечення програмного модуля класифікації елементів анотацій наукових статей ..... 32
3.1 Навчання моделі для класифікації елементів анотацій наукових статей ... 32
3.2 Реалізація програмного модуля .....……..... 39
3.3 Тестування програмного модуля…… 41
Висновки ...……..... 44
Список використаних джерел ….....……. 45
Додаток А Код програми ……... 47
Додаток В Копія опублікованих результатів ……...…... 51

# ВСТУП

Наукові статті – це письмові роботи, в яких дослідники представляють результати своїх наукових досліджень, аналізів або експериментів. Вони зазвичай публікуються в наукових журналах або конференційних збірниках і призначені для обміну знаннями з іншими науковцями та фахівцями в певній галузі.

Обмін висновками через наукові статті має важливе значення для поширення знань і розвитку технологій. Завдяки публікаціям дослідники можуть швидко знайти відповідні релевантні рішення і представити свої результати роботи, також публікації дозволяють науковцям отримувати визнання від своєї спільноти та впливати на напрямки подальших їх досліджень.

Анотація є особливо важливою частиною наукової статті, оскільки вона містить короткий виклад усього дослідження. Цей стислий огляд дає змогу читачам швидко оцінити актуальність і важливість дослідження, не заглиблюючись у повну статтю. Враховуючи величезний обсяг робіт, які публікуються щодня, можливість ефективного перегляду анотацій для виявлення відповідних досліджень є неоціненною. Стандартизований підхід до класифікації речень у анотаціях може значно підвищити ефективність цього процесу.

Обробка природної мови, область штучного інтелекту, зосереджена на взаємодії між комп’ютерами та людською мовою, пропонує значний потенціал для автоматизації аналізу та класифікації тексту. Останні досягнення у цій сфері дозволили комп’ютерам ефективніше розуміти, інтерпретувати та створювати людську мову. Ці технологічні розробки роблять можливим автоматизувати категоризацію речень, тим самим покращуючи доступність і зручність використання наукових досліджень. Використовуючи методи обробки природної мови, дослідники можуть ефективніше орієнтуватися у величезній кількості літератури, сприяючи швидшому й ефективнішому поширенню та застосуванню наукових знань. Тому розробка програмних засобів, що реалізовуватимуть цю можливість є актуальною, а отже, тема кваліфікаційної роботи – актуальна.

Метою кваліфікаційної роботи є створення програмного модуля класифікації

елементів анотацій наукових статей на основі штучного інтелекту. Для досягнення поставленої мети необхідно виконати наступні завдання:

– проаналізувати способи представлення природної мови для комп'ютерних систем та підходи до обробки природної мови засобами штучного інтелекту;

– провести аналіз існуючих рішень та зробити постановку задачі дослідження;

– представити структуру та алгоритмічне забезпечення програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту;

– розробити архітектуру моделі обробки природної мови для виконання класифікації елементів анотацій наукових статей;

– провести навчання моделі обробки природної мови та перевірити точність класифікації;

– розробити програмний модуль та протестувати його.

Об'єктом дослідження є процес класифікації елементів анотацій наукових статей.

Предмет дослідження – програмний модуль класифікації елементів анотацій наукових статей на основі штучного інтелекту.

Для проведення досліджень доцільно провести аналіз існуючих рішень та методів класифікації, чисельний та статистичний аналіз, використати методи теорії машинного навчання і глибоких нейронних мереж для обробки природної мови.

Практичне значення одержаних результатів. В результаті проведеної роботи буде реалізовано програмний модуль класифікації елементів анотацій наукових статей на основі штучного інтелекту

Публікації та апробація кваліфікаційної роботи.

Результати дослідження опубліковано та апробовано в матеріалах міжнародної науково-практичної інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення», 12-13.06.2024 (Додаток Б).

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

## 1.1 Основні аспекти обробки природної мови засобами штучного інтелекту

Обмін результатами в академічних статтях має велике значення для наукового прогресу. Кожна стаття доповнює велику кількість існуючих знань, допомагаючи розвивати різні галузі. Читаючи статті науковці знаходять релевантні рішення, пропозиції та результати, що дозволяє їм покращити їх або створити свої відкриття.

Наукові статті є основним способом, за допомогою якого дослідники діляться своїми висновками з науковою спільнотою. Ці статті не лише повідомляють про нові відкриття, але й сприяють поточним дискусіям у різних сферах.

Розуміння структури наукових статей є важливим як для авторів, так і для читачів, оскільки це допомагає чітко висловлювати та розуміти дослідження [1].

Зазвичай наукова стаття ділиться на кілька ключових розділів:

– анотація представляє короткий підсумок дослідження;

– вступ надає загальну інформацію про статтю та викладає дослідницьке питання або гіпотезу;

– огляд літератури підсумовує наявні дослідження, що стосуються статті;

– методологія описує використані методи дослідження та процедури, порівнює наявні методи із запропонованими;

– результат представляє рішення та результати дослідження;

– висновок узагальнює основні висновки та результати та пропонує майбутні напрямки досліджень.

Серед цих розділів особливо важливою є анотація. Це короткий опис, який показує основні деталі всієї статті. Читаючи анотацію, можна швидко визначити релевантність і значимість дослідження, що робить її важливим елементом для дослідників, яким потрібно ефективно пропрацювати велику кількість статей.

З огляду на велику кількість наукових матеріалів, які публікуються щодня,

дослідникам часто важко швидко переглянути численні анотації, щоб знайти відповідні дослідження. Таким чином, щоб знайти відповідний метод дослідження чи результат, необхідно прочитати всю анотацію до статті. Проблема полягає в тому, що не існує стандартного методу класифікації речень у анотаціях на основі їхніх категорій. Анотації, як правило, містять речення, які належать до однієї з п’яти категорій: ОГЛЯД, МЕТА, МЕТОД, РЕЗУЛЬТАТ та ВИСНОВОК.

Кожна категорія служить певній меті, пояснюючи контекст дослідження, цілі, методів, висновків та наслідків [2].

Поділ анотації на категорії наведені вище дозволяє читачам анотацій ефективніше читати, витрачаючи на кожну анотацію менше часу, а отже читач може ознайомитись з більшим об’ємом інформації за коротший період часу. Також, поділ на такі категорії дозволяє сортувати статті відповідно до кожної категорії. Наприклад обрати усі статті, де мета та результат схожі до бажаного запиту, тоді читач може одразу перейти до опрацювання знайдених статей. Автоматизація цих процесів спростить і прискорить роботу науковців та дослідників.

Обробка природної мови (ОПМ), в англомовній літературі – Natural Language Processing (NLP), – це область штучного інтелекту, зосереджена на взаємодії між машинами (комп’ютерами) та людьми.

Основні завдання ОПМ:

- розпізнавання мовлення (перетворення усного мовлення в текст);
- синтез мовлення (генерація мовлення з тексту);
- аналіз тональності (визначення емоційного забарвлення тексту);
- переклад тексту (автоматичний переклад тексту з однієї мови на іншу);
- токенізація (розбиття тексту на окремі слова або речення);
- розпізнавання сутностей (виявлення та класифікація іменованих об'єктів у тексті);
- витяг інформації (вилучення релевантної інформації з тексту).
- аналіз синтаксису і семантики: визначення граматичної структури та значення тексту.

Протягом останніх років ця сфера досягла значних успіхів, які дозволяють

комп’ютерам розуміти, інтерпретувати та генерувати людську мову, роблячи можливим автоматизацію багатьох задач з аналізу і класифікації тексту.

Людська мова в її звичному для людей вигляді незрозуміла комп’ютерам. Тому слова та речення потрібно перетворити в числове представлення. Цей процес передбачає перетворення тексту в числові вектори, які можуть обробляти комп’ютери. Ці вектори можна створювати як на рівні слів, так і на рівні символів [3, 4].

Процес перетворення слів та речень у числові вектори зазвичай починається з відображення кожного слова в унікальному числовому індексі. Наприклад, розглянемо фразу «сірий кіт». Кожному слову присвоюється унікальне число. Зазвичай числа присвоюються кожному слову в українській мові. Тому ця фраза може мати таке представлення:

– «сірий» = 8896;
– «кіт» = 576.

Отже, фраза «сірий кіт» буде представлена як числовий вектор [8896, 576]. Такий числовий вектор формує основу для подальшої обробки та аналізу, адже комп’ютери можуть опрацювати такий тип даних.

Подібний підхід можна застосувати на рівні символів, присвоївши кожному символу унікальний числовий індекс. Такий підхід використовується у повсякденному розумінні мови у комп’ютерах, як от наприклад формат кодування ASCII чи UTF-8. Наприклад, слово та окличний знак у виразі «кіт!» можна представити як:

– «к» = 23;
– «і» = 28;
– «т» = 30;
– «!» = 65.

Таким чином, «кіт!» буде представлено як числовий вектор [23, 28, 30, 65] на рівні символів. Ці числові вектори дозволяють комп’ютерам обробляти текстові дані структурованим чином, закладаючи основу для більш досконалих методів обробки.

Однак ці числові вектори не охоплюють значення чи зв’язки між словами. Щоб вирішити цю проблему, одним із основоположних досягнень ОПМ стала розробка вбудовування слів. Вбудовування слів — це також числовий вектор, тип представлення слів, який дозволяє представляти слова не тільки як числа в наведеному вище прикладі, а як вектори в багатовимірному векторному просторі. Цей підхід дозволяє фіксувати більше інформації про слово, зокрема семантичні зв’язки між словами на основі їх контексту у реченнях [5].

Наприклад, вектор вбудовування може мати три виміри (для простоти). Фраза «сірий кіт» може бути представлена за допомогою таких векторів:

– «сірий» = [0.1, 0.3, 0.5];
– «кіт» = [0.44, 0.98, 0.1].

Таким чином, фраза «сірий кіт» буде представлена як двовимірний вектор: [[0.1, 0.3, 0.5], [0.44, 0.98, 0.1]].

Ці багатовимірні числові вектори вивчаються під час процесу навчання та охоплюють різні аспекти значень і зв’язків слів. Такі мовні моделі зазвичай вже готові до використання, адже ця модель відображає весь словниковий запас мови. Наприклад, слова які семантично подібні, матимуть вектори, близькі один до одного у векторному просторі [5].

Відомим прикладом вбудованих векторів є зв’язок між «королем» і «королевою». У векторному просторі різниця між числовими векторами «король» і «королева» може охопити концепцію статі. До прикладу, якщо:

– «король» = [0.8, 0.6, 0.2];
– «королева» = [0.8, 0.6, 0.4];
– «чоловік» = [0.7, 0.5, 0.1];
– «жінка» = [0.7, 0.5, 0.3].

Вектор, що представляє «королеву», можна отримати, додавши різницю між «чоловіком» і «жінкою» до вектора «короля».

королева = король - чоловік + жінка

[0.8, 0.6, 0.4] = [0.8, 0.6, 0.2] - [0.7, 0.5, 0.1] + [0.7, 0.5, 0.3]

Цей зв’язок показує, що вбудовування слів може фіксувати складні зв’язки

між словами. Чим більше векторних вимірів, тим більше деталей можна закодувати у слова та словостолучення [6].

Використання вбудовування слів проклало шлях до більш складних мовних моделей, які можуть ефективніше розуміти та обробляти текст. Вбудовування утворюють базовий рівень таких моделей, як мережі довгостроково-короткочасної пам'яті (Long Short-Term Memory networks, LSTM) і двонаправлені мережі довгостроково-короткочасної пам'яті (Bidirectional Long Short-Term Memory networks, BiLSTM), які обробляють послідовні дані та фіксують довготривалі залежності в тексті [7].

LSTM — це тип рекурентної нейронної мережі, призначеної для обробки довгострокових залежностей у послідовних даних. Вони вирішують проблему зникаючого градієнта. Проблема зникаючого градієнта – це явище, яке виникає під час навчання глибоких нейронних мереж, коли градієнти, які використовуються для оновлення вагових коефіцієнтів мережі, стають надзвичайно малими або «зникають», оскільки вони повертаються з вихідних шарів на попередні шари під час тренування. Це веде до того, що слова на початку речення не мають ніякого впливу на контекст слів в кінці речення, що може руйнувати розуміння висловів та речень. LSTM використовує комірку пам'яті для збереження інформації протягом тривалого часу, що вирішує проблему зникаючого градієнта. BiLSTM розширюють можливості LSTM, обробляючи вхідні дані як у прямому, так і в зворотному напрямках, охоплюючи контекст з минулих і майбутніх станів. Вони дозволяють комп'ютерним системам розуміти, що фрази «автобус їде» та «їде автобус» мають однакове значення, навіть якщо числові вектори цих висловів різні [8].

Незважаючи на переваги досягнуті з LSTM та BiLSTM, один із найбільш значних проривів у ОПМ стався з розробкою архітектур транформерів. Трансформери — це тип моделі глибокого навчання, яка базується на механізмах багатоголової уваги для обробки та представлення даних. На відміну від RNN (англ. recurrent neural network), рекурентних нейронних мереж, які обробляють дані послідовно, трансформери можуть обробляти цілі послідовності паралельно, що робить їх більш ефективними для великомасштабних даних. Ця архітектура була

запропонована дослідниками компанії Google в статті 2017 року [9].

Модель двонаправленого енкодера, що базована на основі архітектури трансформерів (BERT) є однією з найпотужніших моделей на основі трансформерів. BERT використовує глибоку двонаправлену трансформерну архітектуру для більш повного розуміння контексту слів у реченні. На відміну від попередніх моделей, які обробляли текст в одному напрямку (зліва направо або справа наліво), BERT обробляє текст в обох напрямках одночасно. Цей двонаправлений підхід дозволяє BERT охоплювати контекст як з лівого, так і з правого боку цільового слова, що веде до глибшого розуміння мови [10].

Ключовою особливістю, яка робить BERT та інші моделі трансформерів такими потужними, є використання механізмів уваги. Увага дозволяє моделі зважити важливість різних слів у реченні під час кодування конкретного слова. Наприклад, у реченні «сірий кіт їсть рибу» механізм уваги може допомогти моделі зрозуміти, що дієслово «їсть» пов’язане з словами «рибу» і «кіт» більше, ніж зі словом «сірий» [10].

Механізми уваги дозволяють моделі зосереджуватися на відповідних частинах вхідної послідовності, що покращує її здатність розуміти контекст і зв’язки в тексті. Ця можливість особливо важлива для фіксації довгострокових залежностей слів у мові. Використовуючи увагу, BERT може ефективно визначати яким словам у реченні слід надати більшого значення, покращуючи його здатність розуміти й опрацьовувати складні мовні структури.

Використання архітертурою BERT вбудованих слів є ще одним ключовим фактором успіху цієї архітектури. Попередньо навчаючись на великій кількості тексту, BERT вивчає високоякісні багатовимірні числові вектори, які вивчають значення та зв’язки між словами. Під час детального налаштування рівнів ці вбудовані функції можна адаптувати до конкретних завдань, що робить BERT надзвичайно універсальним і потужним для різноманітних додатків ОПМ [9, 10].

Здатність глибоко розуміти контекст і точно класифікувати текст робить архітектуру BERT здатним класифікувати речення у анотаціях статей. Навчаючи модель на основі BERT, можна автоматизувати процес класифікації кожного

речення за попередньо визначеними категоріями (ОГЛЯД, МЕТА, МЕТОД, РЕЗУЛЬТАТ та ВИСНОВОК), забезпечуючи точний і ефективний аналіз анотацій.

## 1.2 Огляд і аналіз існуючих рішень

У сфері ОПМ, передові моделі такі як ChatGPT від OpenAI, включаючи GPT- 3.5, і LLaMA (велика мовна модель від Meta), демонструють надзвичайні навички розуміння та написання тексту, який імітує людське мовлення. Ці моделі, які включають BERT і різні версії серії GPT, використовують глибоке навчання для виконання широкого спектру мовних завдань. Такі мовні моделі можуть класифікувати елементи анотацій, проте зазвичай через велику кількість навчальних даних та специфіку виконання завдань для яких вони були створені, такі моделі не здатні проводити класифікацію без перенавчання. Проблема в тому, що такі моделі були навчені для передбачення наступного слова у реченні на основі попередніх слів. Це інший тип задачі, тому запит на точну класифікацію анотації на визначені класи може спричинити неочікуваний результат.

Іншим способом інтерпретації таких потужних мовних моделей є перенавчання під окремий вид задачі, як от класифікація анотацій. Тим не менш, такі сучасні моделі складаються з великої кількості параметрів, що не дозволяє швидко перенавчати моделі під потрібне використання. Проте BERT є відносно невеликою архітектурою, яка містить всього 110 мільйонів параметрів у порівнянні до моделей GPT, кількість параметрів у якій становить від 175 мілярдів параметрів та моделей LLaMA, що містить від 2 до 65 мільярдів параметрів. Потреба в значній обчислювальній потужності для навчання цих моделей на конкретних наборах даних або доменах робить їх дорогими для використання. Архітектура BERT дозволяє використовувати потужні шари своєї архітектури при невеликій кількості параметрів, адже що BERT, що GPT побудовані на основі архітектури трансформерів [9, 10].

Автори [11] досліджують різні методи інтерпретації людської мови для класифікації анотацій статей. Для початку, авторами [11] було використано

класифікатор логістичної регресії з використанням ознак n-gram, витягнутих із речень. Цей метод, простіший і менш ресурсомісткий, ніж підходи нейронних мереж. Автори використовують його як базовий рівень точності в подальших дослідженнях. Вхідні дані для цієї моделі були тільки текстовими, перетвореними в числовий формат за допомогою методу n-gram в діапазоні від уніграм до п’ятиграм, захоплюючи локальний текстовий контекст без будь-якого ширшого семантичного розуміння. Цей метод отримав показник точності F1 у 85,9 на великому наборі даних, що свідчить про його ефективність, незважаючи на простоту.

У ході дослідження [11] автори використовують модель штучної нейронної мережі (ШНМ), яка була використана для створення числових векторів за методом вбудовування слів для речень, що дозволяє класифікувати речення як і на основі контекстного значення поточного речення, так і значень попередніх речень. Ця модель використовує більш складну схему вхідних даних, де кожне речення було перетворене на числовий вектор, що представляє його семантичний зміст та дозволяє мережі враховувати контекстий зв’язок між послідовними реченнями. З використання такого методу, показник точності F1 було підвищено до 88,4, що демонструє важливість контексту у реченнях.

Автори [11] створили модель CRF, яка використовує ознаки n-gram та розглядає послідовність речень у анотації. Цей метод дозволив зафіксувати взаємозалежності між реченнями в анотації, враховуючи як поточний, так і навколишній текст для більш точної класифікації. Використовуючи послідовний характер тексту в анотаціях, ця модель значно підвищує точність показника F1 до 91,5. Ключовим фактором успіху даної моделі є представлення послідовності речень у анотації. Послідовність представлена у виді порядкового числа, яке присвоюється кожному реченні в анотації. Таким чинок порядковий номер класу «ВИСНОВОК» зазвичай вищий ніж у класу «ОГЛЯД». Це дозвояє моделі краще розуміти, що речення класу «ОГЛЯД» знаходить ближче до початку анотації аніж класу «ВИСНОВОК».

Авторами [11] також було представлено модель bi-ANN, що включає

двонаправлений рівень Bi-LSTM з використанням числових векторів вбудовування слів, а також доповнений шаром CRF для оптимізації послідовності речень в анотації. Ця гібридна модель використовує усі переваги методів глибокого навчання та бере до уваги структурну побудову речень завдяки порядковому номеру, обробляючи текст як у прямому, так і в зворотному напрямках, таким чином захоплюючи контекст і підвищуючи точність класифікації. Модель bi-ANN досягла найвищого показника точності F1 91,6, що підкреслює потенціал інтеграції багатьох передових методів для складних завдань класифікації.

Автори [11] провели експерименти моделі на двох вибірках даних. Різниця між ними була у кількості анотацій, велика вибірка розміром 200000 анотацій, та частина великої вибірки у розмірі 20000 анотацій. Попередньо описані резутати були отримані на великій вибірці даних, проте варто зазначити наступні показники отримані авторами на вибірці розміром 20000 анотацій. Класифікатор логістичної регресії з використанням ознак n-gram отримав показник точності F1 83,1. ШНМ продемонструвала більш значне зниження ефективності на меншому наборі даних, з показником точності F1 86,1 порівняно з 88,4 на більшому наборі даних. Цей результат вказує на те, що ШНМ, хоч і потужні, значною мірою залежать від більших наборів даних для захоплення та узагальнення повного спектру мовного апарату. Модель CRF продемонструвала стійкість до зменшення кількості даних з показником F1 89,5 на меншому наборі даних. Показник F1 у 90,0 на меншому наборі даних був отриманий за допомогою моделі bi-ANN. Це вказує на те, що архітектура bi-ANN, яка інтегрує bi-LSTM і рівень CRF, зберігає свою ефективність у класифікації анотацій при меншій кількості даних.

## 1.3 Постановка задачі дослідження

Проведений вище аналіз існуючих рішень і конкретних вимог до завдання класифікації речень у анотація показав, що складні архітектури з використанням структурної інформації про порядок речень дозволяють отримати високу точність класифікації. Послідовність речень у анотації відіграє вирішальну роль у їх

тлумаченні. Кожне речення не лише містить незалежну інформацію, але й семантично пов’язане з попередніми та наступними реченнями.

Отже, для побудови ефективної моделі, необхідно використати глибоке розуміння мови, що можливо за допомогою архітектури BERT у поєднанні з функціями, які фіксують числовий порядок речень і загальну кількість речень в анотації. Такий подвійний підхід гарантує, що класифікатор зосереджено не тільки на змісті речення, а також і на його позиції в анотації.

Необхідно створити комплексну модель, яка буде отримувами на вхід три типи вхідних даних:

– речення;

– числовий порядок речення;

– загальну кількість речень в анотації.

Потрібно провести аналіз існуючих моделей перетворяння слів на багатовимірний числовий вектор як на рівні слів так і на рівні символів, та обрати ефективний метод для використання з архітектурою BERT.

Потрібно навчити таку архітектуру на вибірці даних. Провести аналіз існуючих вибірок, а також використати вибірку з статті [11]. Для початку використати меншу вибірку даних, яка складається з 20000 анотацій для перевірки коректного фунціонування моделі та заощадження комп’ютерних ресурсів.

Представити модель у вигляді модуля, який дозволить отримати анотацію та представити результат із визначеними класами ОГЛЯД, МЕТА, МЕТОД, РЕЗУЛЬТАТ та ВИСНОВОК для кожного із речень анотації.

Метою кваліфікаційної роботи є створення програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту. Для досягнення поставленої мети необхідно виконати такі завдання:

– проаналізувати способи представлення природної мови для комп’ютерних систем та підходи до обробки природної мови засобами штучного інтелекту;

– провести аналіз існуючих рішень для класифікації елементів анотацій наукових статей на основі штучного інтелекту і зробити постановку задачі

дослідження;

– представити структуру програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту;

– розробити архітектуру моделі обробки природної мови для виконання класифікації елементів анотацій наукових статей;

– представити алгоритмічне забезпечення програмного модуля;

– провести навчання моделі обробки природної мови та перевірити точність класифікації;

– розробити програмний модуль та протестувати його.

# 2 СТРУКТУРА ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ ПРОГРАМНОГО МОДУЛЯ КЛАСИФІКАЦІЇ ЕЛЕМЕНТІВ АНОТАЦІЙ НАУКОВИХ СТАТЕЙ

## 2.1 Структура програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту

Програмний модуль класифікації елементів анотацій наукових статей на основі штучного інтелекту складається з моделі обробки природної мови, яка безпосередньо виконує класифікацію, та допоміжних елементів, які дозволяють користувачу використовувати модуль.

Реалізація програмного модуля потребує декількох етапів розробки. Першим етапом є процес навчання моделі ОПМ для виконання класифікації елементів анотацій наукових статей. Другим етапом є створення необхідних програмних умов у вигляді програмного модуля для використання навченої моделі у режимі реального часу.

Ключовим елементом для виконання першого етапу є інформаційне забезпечення моделі ОПМ. Моделі ОПМ навчаються виконувати завдання, визначаючи закономірності та зв’язки в даних, на яких вони тренуються. Без значного та відповідного набору даних ці моделі не можуть ефективно навчатися.

Набір даних PubMed 200k RCT спеціально розроблений для задачі послідовної класифікації елементів наукових статей.

Цей набір даних забезпечує структуроване середовище для навчання моделей методів розпізнавання та класифікації елементів відповідно до їхнього значення в анотації наукової статті. Набір даних містить приблизно 200 000 анотацій із випадково обраних статей, загальна кількість речень в яких становить близько 2,3 мільйона речень [12].

Кожне речення в цих анотаціях ретельно позначено відповідною роллю в анотації. Набір даних структурований так, щоб відображати типовий формат анотацій, таким чином зберігаючи послідовність, яка відображає логічний потік речень у реальній літературі [12].

Запропонована модель ОПМ для класифікації елементів анотації наукових

статей використовує розширені можливості BERT, інтегровані з додатковими контекстними функціями. Правильна підготовка вхідних даних є важливою. Модель використовує складну структуру вхідних даних для покращення її розуміння та точності класифікації [25].

Вхідні дані розділено на три частини:

– текст – головний вид інформації, де модель вивчає лексичний і семантичний вміст речень;

– порядковий номер рядка – кожне речення в анотації буде позначено порядковим номером рядка або позицією, що допомагає моделі зрозуміти порядок інформації, що важливо в тексті з певною структурою, де послідовність може визначати значення;

– загальна кількість рядків – загальна кількість рядків або речень в анотації дозволяє моделі оцінити її обсяг.

Підготовка даних на вхід моделі є важливим кроком у процесі тренування моделі, що в подальшому дозволить використовувати моделі у режимі класифікації.

На рисунку 2.1 зображено структуру програмного модуля класифікації елементів анотації наукових статей на основі штучного інтелекту. Така структура містить вже попередньо навчену модель на першому етапі, яка може використовуватися у режимі класифікації елементів анотації.

На вхід програмного модуля поступає текстовий файл у форматі .txt. Текстовий файл містить анотацію, яку необхідно класифікувати.

Для класифікації елементів анотації необхідно обробити тест для передачі даних на вхід моделі. Для цього необхідно розділити кожне речення. Для кожного з речень потрібно визначити його порядковий номер у тексті. Також необхідно підрахувати кількість речень у анотації та представити всю інформацію про анотацію у вигляді таблиці.

На рисунку 2.2 представлено приклад представлення анотації в результаті виконання модуля підготовки вхідних даних.

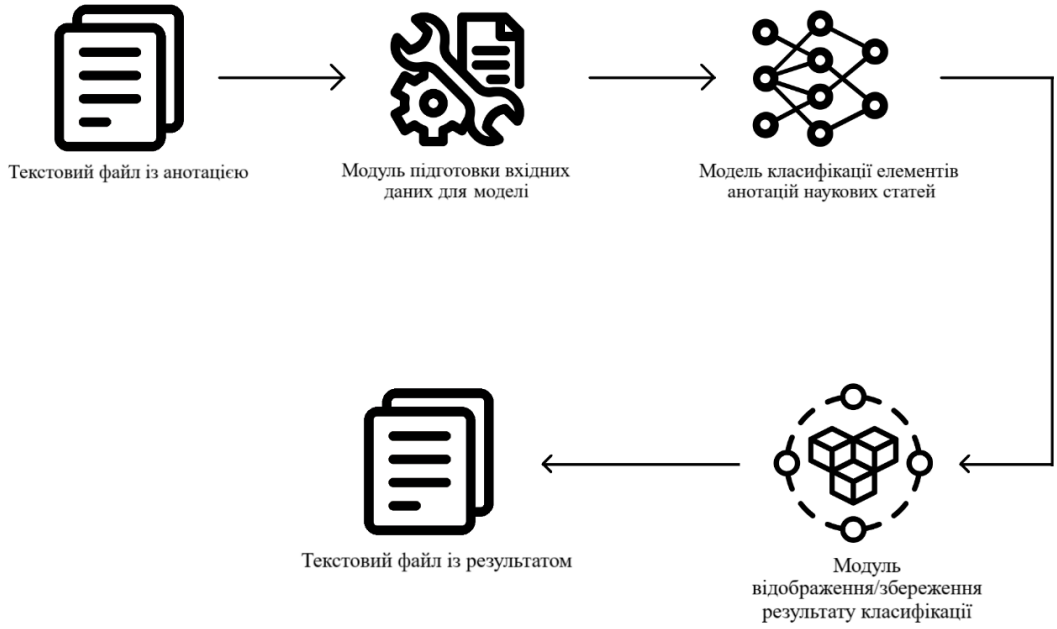


Рисунок 2.1 – Структура програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту

Підготовлені вхідні дані у вигляді трьох окремих змінних використовуються моделлю ОПМ для класифікації кожного з елементів.

| | text | line_number | total_lines |
|---|---|---|---|
| 0 | to investigate the efficacy of @ weeks of dail... | 0 | 11 |
| 1 | a total of @ patients with primary knee oa wer... | 1 | 11 |
| 2 | outcome measures included pain reduction and i... | 2 | 11 |
| 3 | pain was assessed using the visual analog pain... | 3 | 11 |
| 4 | secondary outcome measures included the wester... | 4 | 11 |

Рисунок 2.2 – Приклад представлення анотації в результаті виконання модуля підготовки вхідних даних

В результаті роботи моделі для кожного елементу присвоюється один з класів ОГЛЯД, МЕТА, МЕТОД, РЕЗУЛЬТАТ та ВИСНОВОК. Далі результат класифікації представлено у вигляді тексту та збережено у файл формату .txt. Результати класифікації також представлені користувачу для швидкого опрацювання анотації.

## 2.2 Архітектура моделі ОПМ на основі BERT із множинним вводом даних

На рисунку 2.3 представлена архітектура моделі ОПМ на основі BERT із трьома векторами вхідних даних. Складна архітектура моделі, розроблена для класифікації тексту шляхом використання як текстових функцій, так і структурної інформації елементів анотацій.

На рисунку зображено три головні гілки інформації, проте ця архітектура містить чотири типи вхідних даних. Вхідні рівні token_input та mask_input це похідні змінні тексту, які отримуються в результаті використання токенізатора.

В обробці природної мови токенізація є критичним етапом, який передбачає розбиття тексту на менші одиниці, які називаються токенами. Цей процес важливий для підготовки текстових даних для навчання моделей глибокого навчання, таких як BERT.

Для токенізації тексту необхідно використати AutoTokenizer з бібліотеки трансформетір Hugging Face, спеціально навчений на моделі «bert-base-uncased», щоб токенізувати вхідні речення [13]. Кожне речення кодується за допомогою методу encode_plus, який виконує кілька важливих функцій:

– максимальна довжина гарантує, що всі токенізовані виходи доповнюються або скорочуються до узгодженої довжини у 55 токенів, що важливо для блокової обробки даних в моделях глибокого навчання;

– заповнення застосовує пусті токени (імітацію слів), щоб гарантувати, що всі закодовані речення досягають максимальної вказаної довжини, сприяючи рівномірному розміру вхідної інформації;

– спеціальні токени додають маркери, такі як [CLS] (початок послідовності) і [SEP] (розділювач), які потрібні BERT для розуміння меж і сегментів у текстах;

– маска уваги створює векторне представлення, щоб розрізняти фактичні маркери та заповнення (імітацію), ефективно керуючи механізмом уваги моделі;

– перетворення на тензор – токенізований вихід (вхідні ідентифікатори та маски уваги) перетворюються на тензори TensorFlow. Це перетворення необхідне

для сумісності з операціями TensorFlow і для використання прискорення GPU під час навчання моделі [14].

Окрім текстової вхідної змінної, модель містить дві числові змінні введення – номер рядка та загальну кількість рядків у кожній анотації. Ці функції забезпечують структурний контекст моделі, що має вирішальне значення для завдань, пов'язаних із послідовними даними. Щоб забезпечити придатність цих числових даних для навчання моделі необхідно застосувати одноразове кодування до цих числових вхідних даних, перетворюючи їх на двійкову матрицю. Цей метод допомагає моделі інтерпретувати ці характеристики, не припускаючи природного порядкового зв'язку між різними числами.

Вхідні шари:

– token_input: цей рівень опрацьовує вхід токенізованого тексту, послідовності закодованих слів, які будуть оброблені моделлю BERT;

– mask_input: супроводжує токенізований текст, щоб вказати моделі BERT, які частини вхідних даних є фактичними даними, а які – імітованими;

– line_number_input: отримує порядковий номер рядка кожного речення як вхідні дані, допомагаючи моделі зрозуміти позицію речення в анотації;

– total_lines_input: бере загальну кількість рядків у анотації, надаючи контекст про довжину та структуру анотації.

tf_bert_model (TFBertModel) це модель на основі архітектури трансформера, попередньо навчена на великій вибірці даних [15]. Цей рівень обробляє токенізовані вхідні дані, щоб створити контекстне представлення кожного токена. Це і є процес створення вбудованих числових векторів у багатовимірному векторному просторі, які було описано у підрозділі 1.1.

Багатовимірні числові вектори фіксують як значення окремих слів, так і їхній контекст у реченні.

global_max_pooling1d_1 (GlobalMaxPooling1D) – рівень, що зменшує розмірність вихідного вектора з моделі BERT, взявши до уваги максимальне значення у векторі.

Ця операція витягує найважливіші характеристики з вбудованих векторів, які

потім передаються на наступні рівні.

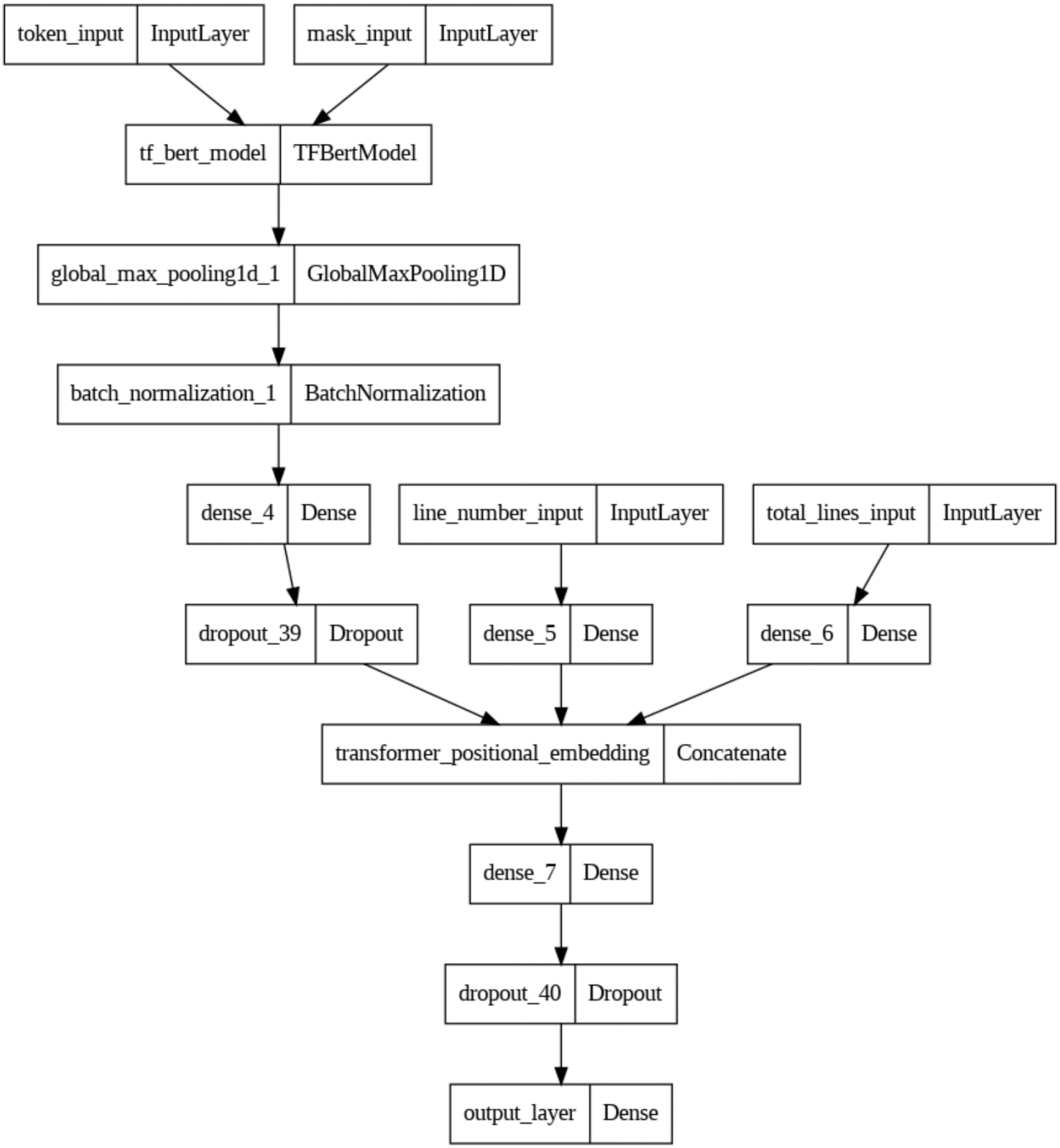


Рисунок 2.3 – Архітектура моделі ОПМ на основі BERT із трьома векторами вхідних даних

batch_normalization_1 (BatchNormalization) нормалізує активації з рівня об’єднання, стабілізуючи та прискорюючи процес навчання, підтримуючи середнє вихідне значення близько до нуля і вихідне стандартне відхилення близько до

одиниці.

dense_4, dense_5, dense_6, dense_7 (Dense) це звичайні шари нейронної мережі, які змінюють вагові коефіцієнти під час навчання, щоб встановити зв'язок між вхідними та вихідними даних рівня. Кожен рівень має різну роль залежно від свого положення в мережі. dense_4 обробляє вектори з рівня batch_normalization_1, тоді як dense_5 і dense_6 обробляє вектори з представлення порядкового номера рядка та загальної кількості рядків відповідно. dense_7 об’єднує та обробляє виходи, які були об’єднані.

dropout_39, dropout_40 (Dropout) – шари випадковим чином прирівнюють частину вхідних векторів до 0 під час кожної епохи навчання, що допомагає запобігти перенавчанню (overfitting).

transformer_positional_embedding (Concatenate) це рівень який об’єднує вихідні дані dense_5 і dense_6 із векторними результатами моделі BERT, об’єднуючи вивчені ознаки тексту зі структурною інформацію.

output_layer (Dense) це останній рівень моделі, який також має функцію активації, що підходить для завдання класифікації (наприклад, softmax для багатокласової класифікації). Цей рівень виводить розподіл ймовірностей за класами, забезпечуючи остаточну класифікацію речень [16].

Запропонована архітектура є складною, тому в результаті об’єднання всіх рівнів вона складається із 109610277 параметрів. Велика частина цих параметрів належать моделі BERT.

Для задачі багатокласової класифікації, де метою є класифікація тексту за кількома попередньо визначеними категоріями, стандартною практикою є використання функції активації softmax у вихідному рівні. Функція softmax перетворює логіти (вихідні дані моделі до активації) у ймовірності, беручи експоненту кожного виходу, а потім нормалізуючи ці значення шляхом ділення на суму всіх експонент. Це забезпечує розподіл ймовірностей між цільовими класами [17].

У завданнях класифікації, особливо з незбалансованими наборами даних, точність класифікації не відображає реальний результат. Наприклад, у наборі

даних, де 90% даних належать до одного класу, модель, яка наївно прогнозує цей основний клас для всіх вхідних даних, досягне 90% точності, незважаючи на відсутність прогнозувань для меншого класу.

Отже, точність не завжди є відповідним показником. Показник F1 використовується як більш надійний показник. Показник F1 обчислюється за наступною формулою:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}, \quad (2.1)$$

де *F1* – показник F1;

*precision* – відношення правильно прогнозованих «позитивних» спостережень до загальної кількості прогнозованих «позитивних» результатів;

*recall* – відношення правильно прогнозованих «позитивних» спостережень до всіх фактичних «позитивних» результатів.

Показник F1 врівноважує precision і recall моделі, роблячи його кращим показником неправильно класифікованих випадків, ніж показник точності. Це корисно, коли розподіл класів є нерівномірним [18].

Під час навчання моделі було використано показник точності для налагодження вагових коефіцієнтів.

## 2.3 Алгоритмічне забезпечення програмного модуля

На рисунку 2.4 представлено алгоритм роботи програмного модуля класифікації елементів анотації наукових статей на основі штучного інтелекту [25]. Першим кроком алгоритму є ввід анотації у програмний модуль, а саме: текстовий файл записується у змінну програмного модуля.

Наступним кроком анотація поділяється на речення. Кожне незалежне речення проходить через ідентифікацію та заміну спеціальних символів. Під спеціальними символами мається на увазі розділові знаки, числа, елементи обчислення та формул, а також інші символи, які не є частиною мови.

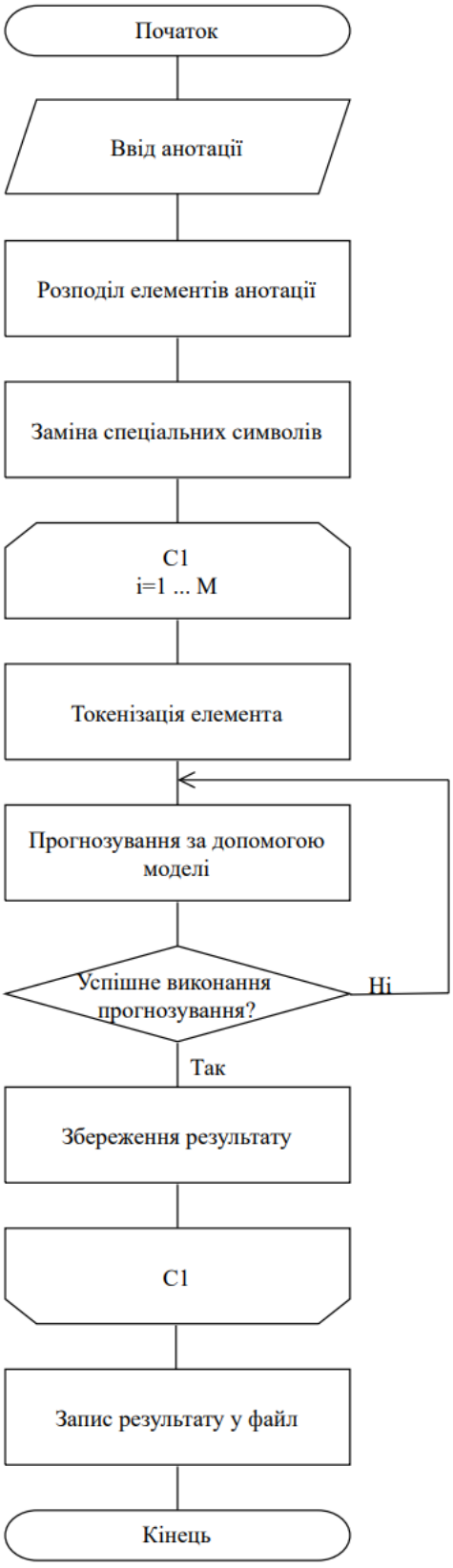


Рисунок 2.4 – Схема алгоритму роботи програмного модуля класифікації елементів анотації наукових статей на основі штучного інтелекту

Алгоритм ініціює змінну, яка контролює кількість ітерацій у циклі. Змінна

«i» представляє поточну ітерацію, «M» відображає кількість елементів (речень) у анотації статті. «i»-те речення у анотації проходить процес токенізації під час кожної ітерації. Представлене у вигляді тензора речення стає повністю готовим до подачі на вхід моделі.

Процес прогнозування за допомогою моделі є ресурсозатратним, він займає більше часу у порівнянні з іншими процесами алгоритму.

Якщо прогнозування відбулося успішно, то результат зберігається у змінну, а потім програмний модуль переходить до іншої ітерації та виконує прогнозування для всіх інших ітерацій.

Після виконання прогнозування для всіх елементів, тобто виконавши усі ітерації, програмний модуль записує результат класифікації у текстовий файл.

На рисунку 2.5 представлено алгоритм навчання класифікатора ОПМ на основі архітектури BERT. Так як архітектура запропонованої моделі є великою, кількість вільної пам’яті у графічному процесорі комп’ютерної системи є важливим елементом навчання. Для того щоб помістити модель та пакети даних у пам'ять, необхідно розділити вибірку даних на пакети.

У випадку класифікатора ОПМ пакети даних містять 16 елементів анотації. Елементи пакета є випадково обраними з набору даних, тож малоймовірно що в одному пакеті можуть бути елементи однієї анотації. Такий випадковий порядок дозволяє моделі розуміти зміст конкретного елемента без прив’язки до попередніх речень в анотації. Це дозволяє моделі краще класифікувати елементи в подальшому на нових зразках анотацій.

Одна епоха навчання – це коли модель опрацювала всі пакети даних, тобто весь набір даних.

Навчання в епохах дозволяє моделі ітеративно вивчати весь набір даних. Кожна епоха представляє повний прохід через увесь набір даних, що дає моделі численні можливості вивчати закономірності та зменшувати помилки з часом. Дані часто можуть надходити з властивим порядком (наприклад, відсортовані за класами, згруповані за ознаками), що може привести до упередженого навчання, якщо модель бачить лише певні шаблони в послідовності під час навчання.

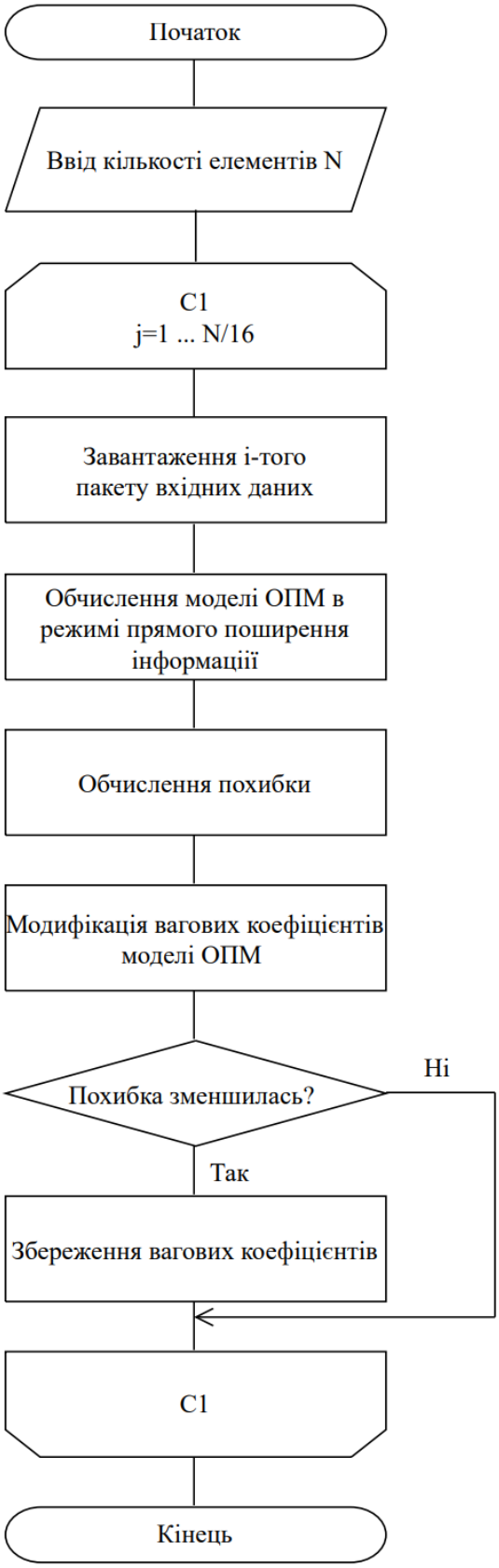


Рисунок 2.5 – Схема алгоритму навчання класифікатора ОПМ на основі архітектури BERT

Перетасування (зміна порядку розміщення даних у вибірці) допомагає

рандомізувати точки даних, таким чином зменшуючи ймовірність того, що модель запам’ятає певні шаблони, що залежать від порядку, і замість цього навчиться краще узагальнювати всі точки даних.

Алгоритм розпочинається з ініціалізації змінних:

– змінна «j» представляє поточну ітерацію;

– «N» відображає кількість елементів анотації у навчальній вибірці, а «N/16» вказує на кількість пакетів даних у навчальній вибірці.

Ітерація циклу містить у собі процеси завантаження відповідного пакету даних, обчислення параметрів моделі, обчислення похибки, модифікацію вагових коефіцієнтів та у разі зменшення похибки, зберігає вагові коефіцієнти.

Після закінчення опрацювання всіх пакетів даних, алгоритм пройшов одну епоху тренування. По закінченні тренування декількох епох, модель готова до використання у програмному модулі класифікації елементів анотацій наукових статей на основі штучного інтелекту.

# 3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ПРОГРАМНОГО МОДУЛЯ КЛАСИФІКАЦІЇ ЕЛЕМЕНТІВ АНОТАЦІЙ НАУКОВИХ СТАТЕЙ

## 3.1 Навчання моделі для класифікації елементів анотацій наукових статей

Процес навчання запропонованої моделі обробки природної мови складається із чітко визначених кроків, від підготовки даних до навчання моделі та оцінювання. Кожен крок розроблено для того, щоб гарантувати ефективне розуміння моделлю як текстових, так і контекстних елементів набору даних.

Підготовка даних є основоположним кроком, на якому необроблені текстові дані перетворюються в структурований формат, який може ефективно оброблятися моделлю.

Функція preprocess_text_with_line_numbers, зображена на рисунку 3.1, використовується для читання всього файлу як списку рядків. Це попередній крок для подальшої обробки даних. Ця функція обробляє кожен рядок із набору даних. Ідентифікує та розділяє окремі анотації на основі певних маркерів (наприклад, ### для нової анотації). Для кожної анотації функція розбиває текст на окремі речення, призначає номери рядків і обчислює загальну кількість рядків, інкапсулюючи цю інформацію в структурований формат (словник).

```python
# Function that extracts data from text
def preprocess_text_with_line_numbers(filename):
    """
    Returns a list of dictionaries of abstract line data containing .
    """
    input_lines = get_lines(filename=filename)
    abstract_lines = ""
    abstract_samples = []
    # loop through each line in the target file
    for line in input_lines:
        if line.startswith('###'):
            abstract_id = line
            abstract_lines = ""
        elif line.isspace():
            abstact_line_split = abstract_lines.splitlines()
            for abstract_line_number, abstract_line in enumerate(abstact_line_split):
                line_data = {}
                target_text_split = abstract_line.split("\t")
                line_data["target"] = target_text_split[0]
                line_data["text"] = target_text_split[1].lower()
                line_data["line_number"] = abstract_line_number
                line_data["total_lines"] = len(abstact_line_split) - 1
                abstract_samples.append(line_data)
        else:
            abstract_lines += line
    return abstract_samples
```

Рисунок 3.1 – Функція preprocess_text_with_line_numbers

Після попередньої обробки дані потрібно відформатувати у відповідну форму для навчання моделі.

Кожен попередньо оброблений зразок перетворюється на DataFrame для полегшення маніпулювання та доступу даних. Потім речення вилучаються в списки для подальшої обробки, наприклад, токенізації.

Категоричні (якісні) дані, такі як цільові класи, перетворюються у однократно закодований формат, який є двійковим матричним представленням класів. Це кодування необхідне для виконання моделлю багатокласової класифікації.

Важливим етапом підготовки даних є токенізація тексту. У процесі навчання було використвно токенізатор BERT, розроблений для ефективної обробки складних завдань. На рисунку 3.2 зображено функцію яка відповідає за процес токенізації.

```
def prepare_tokens_t(sentences):
  encoded_sentences = []
  encoded_masks = []
  for i in sentences:
    encoded_sentence = tokenizer_t.encode_plus(i, max_length=55, truncation=True,
                                               padding="max_length",
                                               add_special_tokens=True,
                                               return_token_type_ids=False,
                                               return_attention_mask=True,
                                               return_tensors='tf')
    encoded_sentences.append(encoded_sentence['input_ids'].numpy().tolist()[0])
    encoded_masks.append(encoded_sentence['attention_mask'].numpy().tolist()[0])
  encoded_sentences = tf.convert_to_tensor(encoded_sentences, np.int32)
  print(encoded_sentences.shape)
  encoded_masks = tf.convert_to_tensor(encoded_masks, np.int32)
  print(encoded_masks.shape)
  return encoded_sentences, encoded_masks
```

Рисунок 3.2 – Функція prepare_tokens_t

Параметри, які використовуються в методі encode_plus токенізера BERT, і їхнє значення було описано у підрозділі 2.2.

Після токенізації дані необхідно представити у форматі tf.data.Dataset, який використовується для побудови високопродуктивного складного конвеєра

введення даних. Це пришвидшує процес навчання, особливо під час навчання на великій кількості даних.

На рисунку 3.3 зображено функцію, яка будує архітектуру запропонованої моделі з стандартних рівнів присутніх у бібліотеці Tensorflow.

```
# Custom BERT model

# 1. Setup token inputs/model
token_inputs = layers.Input(shape=(55,), dtype=tf.int32, name="token_input")
mask_inputs = layers.Input(shape=(55,), dtype=tf.int32, name="mask_input")

embeddings = bert(token_inputs, mask_inputs)[0] #3D tensor without pooling
bert_layer = layers.GlobalMaxPool1D()(embeddings)
bert_layer = layers.BatchNormalization()(bert_layer)
last_bert_layer = layers.Dense(128, activation='relu')(bert_layer)
last_bert_layer = layers.Dropout(0.2)(last_bert_layer)
bert_model = tf.keras.Model(inputs=[token_inputs, mask_inputs], outputs=last_bert_layer)

# 3. Line number inputs model
line_number_inputs = layers.Input(shape=(15,), dtype = tf.float32, name="line_number_input")
x = layers.Dense(32, activation='relu')(line_number_inputs)
line_number_model = tf.keras.Model(line_number_inputs, x)

# 4. Total lines model
total_lines_inputs = layers.Input(shape=(20,), dtype = tf.float32, name="total_lines_input")
y = layers.Dense(32, activation='relu')(total_lines_inputs)
total_line_model = tf.keras.Model(total_lines_inputs, y)

# 5. Combine positional embeddings with combined token and char embeddings
tribrid_embeddings = layers.Concatenate(name="transformer_positional_embedding")([line_number_model.output,
                                                                                  total_line_model.output,
                                                                                  bert_model.output])
final_dense = layers.Dense(128, activation="relu")(tribrid_embeddings)
final_dense = layers.Dropout(0.5)(final_dense)

# 6. Create output layer
output_layer = layers.Dense(5, activation="softmax", name = "output_layer")(final_dense)

# 7. Put all together
model_bert_embed = tf.keras.Model(inputs=[line_number_model.input,
                                          total_line_model.input,
                                          token_inputs,
                                          mask_inputs], outputs = output_layer, name="model_3")

model_bert_embed.layers[2].trainable=False
```

Рисунок 3.3 – Функція побудови архітектури моделі

Методом послідовного об’єднання шарів, вхід та вихід кожного з рівнів нейронної мережі з’єднано для передачі даних через параметри різних рівнів. Варто приділити увагу рівню під назвою output_layer. Цей рівень є останнім в архітектурі та використовує функцію активації softmax для перетворення

результатів у показники ймовірностей для кожного з п’яти класів.

Після визначення архітектури моделі наступними критичними кроками є компіляція та навчання, які необхідні для перетворення теоретичної моделі на практичний інструмент.

Модель скомпільовано за допомогою функцій TensorFlow із такими налаштуваннями:

– CategoricalCrossentropy вибрано через її придатність для обробки задач багатокласової класифікації, де кожен вхідний зразок має бути класифікований в одному з кількох класів;

– оптимізатор Adam використовується через його ефективність у обробці великих наборів даних і складних архітектур нейронних мереж. Він динамічно регулює швидкість навчання, що корисно для моделей із великими та різними градієнтами, як це зазвичай буває в моделях глибокого навчання, таких як BERT;

– точність є загальною метрикою, вона може не відображати реальної картини ефективності моделі через можливий дисбаланс класів у даних. Таким чином, точність використовується для оновлення вагових коефіцієнтів, але не є виключною оцінкою ефективності [19].

У машинному навчанні набори даних для навчання, тестування та перевірки відіграють вирішальну роль у розробці та оцінці моделей. Ці набори даних використовуються для забезпечення того, щоб модель машинного навчання не тільки добре навчалася, але й добре узагальнювала нові, невідомі дані.

Навчальний набір даних — це основний набір даних, який використовується для навчання моделі машинного навчання. Цей набір даних використовується на етапі навчання, коли модель вчиться визначати закономірності, приймати рішення або створювати результати на основі вхідних характеристик. Як правило, більший і різноманітніший набір навчальних даних призводить до моделі, яка може вивчати більш повні шаблони і, отже, краще працювати з реальними даними.

Коли модель навчена на навчальних даних, дуже важливо налаштувати її параметри та вибрати найкращу версію моделі. Для цього використовується набір даних перевірки, щоб допомогти в оцінці моделі під час процесу навчання,

дозволяючи точно налаштувати гіперпараметри моделі. Цей набір даних допомагає запобігти перенавчанню моделі.

Тестовий набір даних — це окрема частина даних, яка використовується для оцінки продуктивності повністю навченої моделі. Вкрай важливо, щоб ці дані ніколи не використовувалися на етапах навчання чи перевірки. Тестові дані дають остаточну оцінку того, наскільки добре модель буде працювати на нових даних. Він використовується лише після повного навчання та перевірки моделі. Показники, отримані з тестового набору даних (такі як точність, показник F1 тощо), забезпечують неупереджену оцінку кінцевої моделі, оскільки вони вказують на те, як модель працюватиме в реальному сценарії на основі даних, яких модель не бачила [20].

Процес навчання запускається за допомогою функції представленої на рисунку 3.4. Фрагменти коду програми навчання моделі представлені у додатку А.

```
history_bert_embed=model_bert_embed.fit(train_tr_token_pos_dataset,
                    steps_per_epoch=(len(train_tr_token_pos_dataset)),
                    epochs=15,
                    validation_data=val_tr_token_pos_dataset,
                    validation_steps=(len(val_tr_token_pos_dataset)),
                    callbacks=[checkpoint_callback, reduce_lr])
```

Рисунок 3.4 – Функція запуску навчання моделі

Навчальні дані групуються, що означає, що вони розділені на підмножини, які модель обробляє ітеративно. Цей метод необхідний для керування ресурсами пам’яті та зазвичай призводить до швидшої обробки порівняно з подачею даних по одному зразку на раз.

Припасування даних використовується для підготовки даних під час навчання моделі на поточному пакеті. Це допомагає зменшити час, протягом якого модель має чекати отримання нових даних для обробки, тим самим прискорюючи навчання. Контрольні точки моделі зберігають найкращу(показник точності) модель під час навчання на основі точності на вибірці перевірки. Це гарантує

збереження найефективнішої моделі, навіть якщо продуктивність моделі погіршується в наступні епохи.

Модель адаптується до навчальних даних за допомогою методу підгонки, який навчає модель під час фіксованої кількості епох на наданому наборі даних. Навчальні та перевірочні набори даних використовуються щоб дати змогу моделі не лише вивчати основні закономірності в навчальних даних, але й добре узагальнювати нові, невідомі дані.

На рисунку 3.5 зображено графік зміни похибки моделі від час тренування. На вертикальній осі представлено значення похибки, на горизонтальній осі представлено номер епохи тренування. Синя крива відображає графік зміни похибки на вибірці навчання, оранжева крива відображає вибірку перевірки. Як видно на графіку, похибка моделі зменшується протягом навчання моделі, це означає що модель покращує розуміння даних на обох вибірках даних.

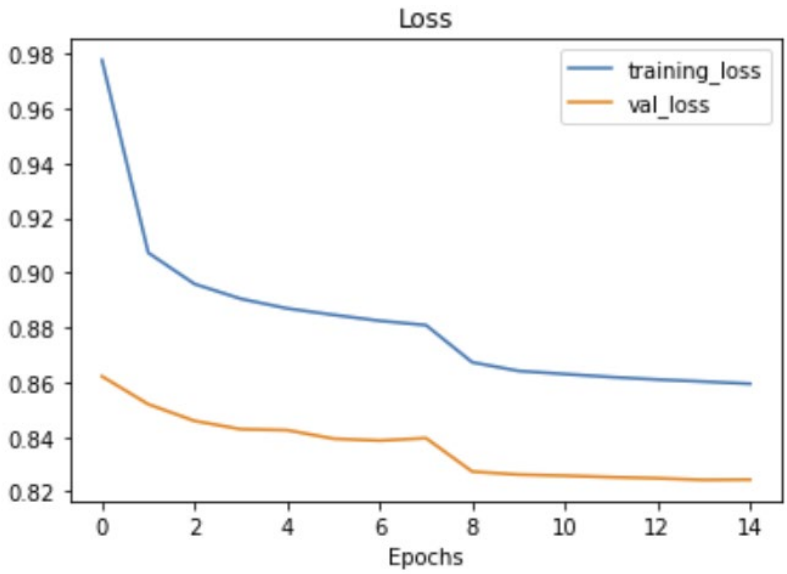


Рисунок 3.5 – Графік зміни похибки моделі

На рисунку 3.6 зображено графік зміни точності моделі під час тренування. На вертикальній осі представлено значення точності, на горизонтальній осі представлено номер епохи тренування. Синя крива відображає графік зміни похибки на вибірці навчання, оранжева крива відображає вибірку перевірки. Як видно на графіку, точність моделі збільшується протягом навчання моделі на обох

наборах даних, це означає що модель не перенавчена до набору тренування та буде добре працювати на нових даних. На останній епосі точність на вибірці перевірки досягає показника 0.89, що є дуже високим показником для моделі ОПМ.

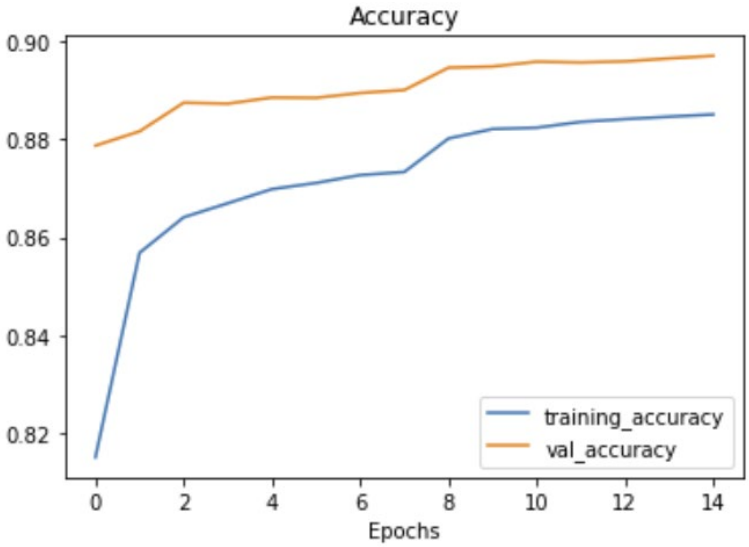


Рисунок 3.6 – Графік зміни точності моделі

Після навчання моделі та припасування параметрів за допомогою вибірки перевірки, було перевірено результат моделі на вибірці тестування. На рисунку 3.7 зображено результати моделі. Головним показником є показник F1 який наблизився до значення 0.89. Це високий показник, що означає високу ефективність моделі. Така модель здатна рівномірно класифікувати елементи анотації. Схожі значення точності на вибірці тестування, перевірки та навчання показують здатність моделі працювати на різних незалежних вибірках даних. Це означає що модель, під час роботи на реальних даних буде виконувати класифікацію з такою ж ефективністю.

```
{'accuracy': 89.18533266965323,
 'precision': 0.8941773037650746,
 'recall': 0.8918533266965323,
 'f1': 0.8889338781440873}
```

Рисунок 3.7 – Результати моделі на вибірці тестування

## 3.2 Реалізація програмного модуля

Для реалізації програмного модуля та навчання моделі було використано мову програмування Python, бібліотеку машинного навчання Tensorflow, середовище Visual Studio Code та середовище керування версіями бібліотек Anaconda [25].

Python – це універсальна мова програмування, яка широко використовується в сфері машинного навчання та науки про дані завдяки своїй простоті та читабельності в поєднанні з потужними бібліотеками та фреймворками. Синтаксис Python зрозумілий, що робить його доступним та достатньо надійним для створення складних програм. Для завдань машинного навчання Python пропонує різні бібліотеки, такі як NumPy для числових операцій, pandas для маніпулювання даними та Matplotlib для візуалізації даних, що робить його незамінним інструментом для дослідників і розробників у спільноті науки про дані [21].

TensorFlow – це бібліотека машинного навчання з відкритим вихідним кодом, розроблена Google, і це одна з найпопулярніших платформ, які використовуються для створення та навчання моделей машинного навчання. TensorFlow надає комплексну, гнучку екосистему інструментів, бібліотек і ресурсів спільноти, яка дозволяє дослідникам просувати найсучасніші технології машинного навчання, а розробникам легко створювати й розгортати програми на базі машинного навчання. TensorFlow чудово справляється з великомасштабними чисельними обчисленнями, що важливо для навчання складних нейронних мереж. Його здатність виконувати автоматичну диференціацію та розгортати обчислення на кількох центральних або графічних процесорах робить його високоефективним інструментом для завдань машинного навчання [22].

Для проектів машинного навчання VS Code пропонує відмінну підтримку для розробки Python із розширеннями, які надають такі потужні функції, як IntelliSense (завершення коду), навігація по коду та розширені можливості налагодження [23].

Anaconda – це середовище із відкритим вихідним кодом мов програмування Python і R для наукових обчислень, метою якого є спрощення керування пакетами

та версіями бібліотек. Дистрибутив Anaconda містить велику кількість попередньо встановлених бібліотек обробки даних і машинного навчання, що робить його ідеальною платформою для академічних і дослідницьких цілей [24].

На рисунку 3.8 зображено приклад текстового файлу який подається на вхід програмного модуля.

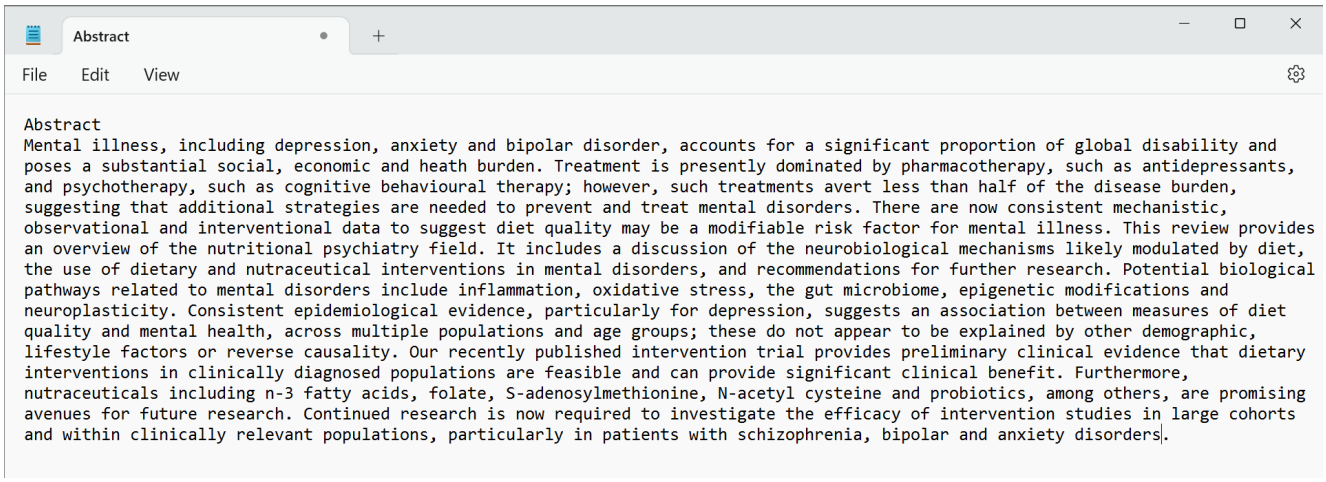


Рисунок 3.8 – Приклад текстового файлу анотації

Для використання програмного модуля необхідно використати команду зображену на рисунку 3.9 у терміналі. Команда запускає програмний код, який використовує навчену модель для класифікації елементів анотації.

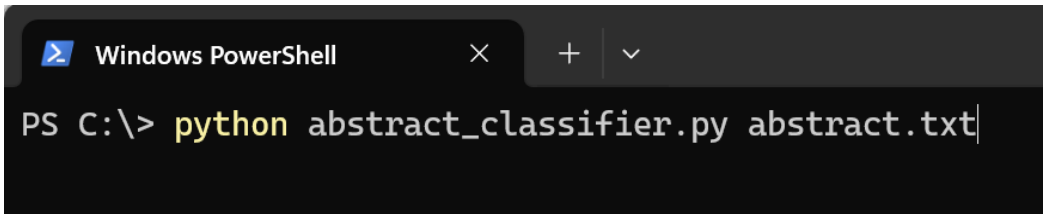


Рисунок 3.9 – Приклад використання програмного модуля

Після успішного виконання класифікації, файл abstract.txt було змінено результатом класифікації. Приклад класифікації елементів анотації зображено на рисунку 3.10.

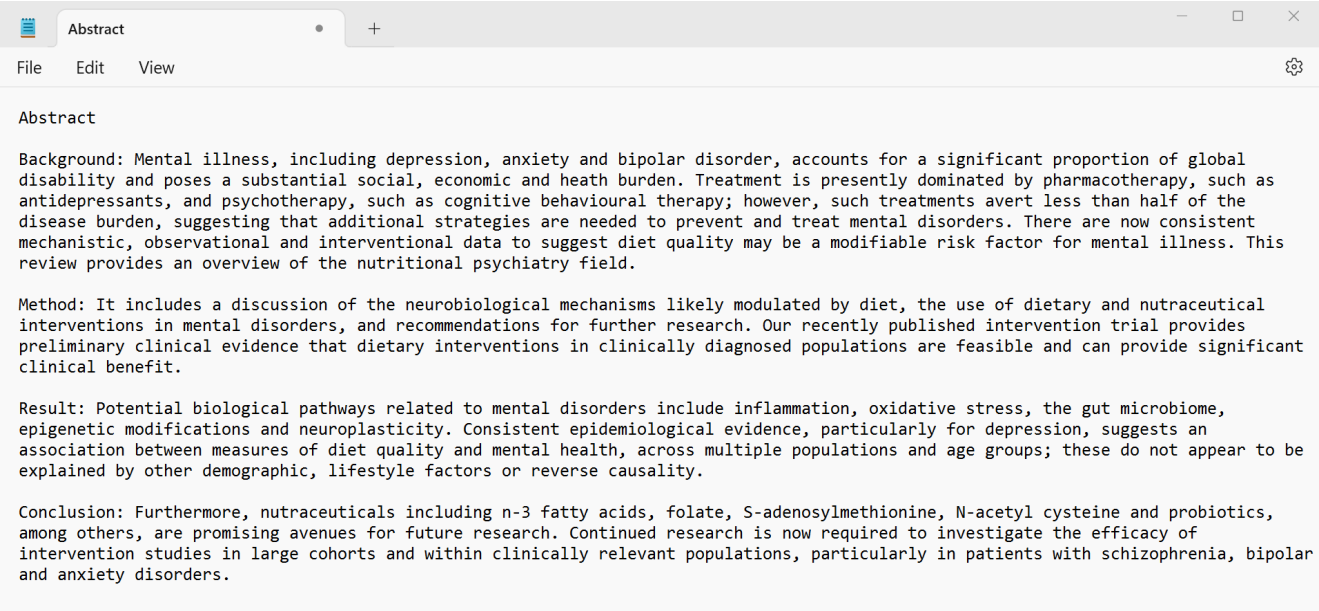


Рисунок 3.10 – Результат класифікації елементів анотації

Таким чином, в результаті класифікації читач може сконцентруватися на необхідних елементах дослідження та перейти до опрацювання важливих деталей.

## 3.3 Тестування програмного модуля

Тестування є важливим етапом у розробці будь-якого програмного модуля, особливо для модуля, який містить модель машинного навчання. Ретельне тестування гарантує, що модуль не тільки відповідає встановленим для нього функціональним вимогам, але й справляється з помилками, підтримує стабільність за різних умов і забезпечує надійні та послідовні результати. Для модуля машинного навчання тестування є важливим, оскільки воно гарантує, що і код, і модель працюють належним чином за різноманітних сценаріїв, які можуть виникнути в реальних умовах.

Тестування моделі має вирішальне значення для ефективності та надійності програмного модуля. Результати тестування ефективності моделі було описано в підрозділі 3.1. Для загального тестування роботи програмного модуля було створено три поширені сценарії виконання коду.

Перший тестовий сценарій розглядає ситуацію, коли вхідний файл, вказаний

користувачем, не існує. Очікувана поведінка модуля є перевірка існування файлу за вказаним шляхом перед спробою читання з нього. Якщо файл не знайдено, модуль має повернути чітке та інформативне повідомлення про помилку, наприклад «Помилка: указаний файл не існує. Перевірте шлях до файлу та повторіть спробу». На рисунку 3.11 зображено повідомлення про відсутність файлу.

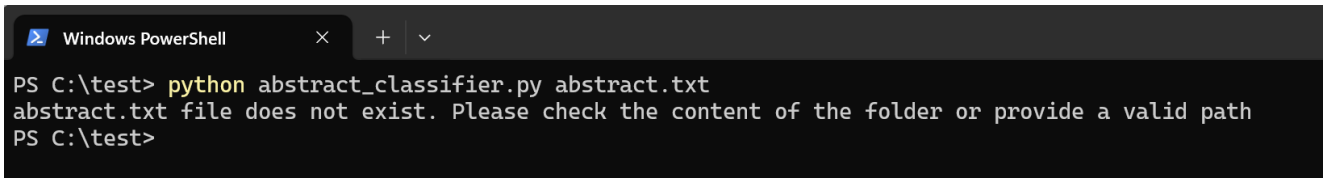


Рисунок 3.11 – Повідомлення про відсутність файлу

Цей тест гарантує, що користувачі відразу знають про проблему відсутності файлу, що може запобігти плутанині та переконатися, що користувачі знають, що їхні введені дані потребують виправлення.

Другий тестовий сценарій розглядає ситуацію коли файл існує, але він не містить анотацій для обробки. Після виявлення порожнього файлу або файлу без очікуваного вмісту модуль має повернути повідомлення про помилку: «Помилка: файл порожній або не містить анотацій». Тестування порожніх файлів гарантує, що модуль може обробляти різні вхідні дані та інформувати користувача про конкретну проблему з наданими даними. На рисунку 3.12 представлено результат такого сценарію.

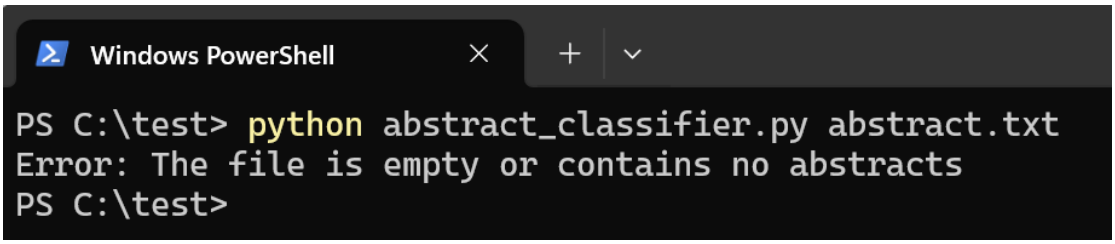


Рисунок 3.12 – Повідомлення про відсутність вмісту у файлі

Третій сценарій перевіряє реакцію модуля на помилки під час завантаження

або фази ініціалізації моделі машинного навчання, наприклад, пошкоджені файли моделі або несумісні версії бібліотеки.

Модуль має спробувати завантажити необхідну модель машинного навчання під час запуску. Якщо під час цього процесу сталася помилка через відсутність файлів, пошкодження чи будь-яку іншу проблему, модуль має виявити цю помилку та надати чітке повідомлення, наприклад «Помилка: виникла проблема із завантаженням моделі. Переконайтеся, що файл моделі правильний і спробуйте ще раз.", що зображено на рисунку 3.13.

```
PS C:\test> python abstract_classifier.py abstract.txt
Error: There was a problem loading the model. Make sure the model file is correct and try again
PS C:\test>
```

Рисунок 3.13 – Повідомлення про помилку при завантаженні моделі

Такий тест має вирішальне значення для того, щоб переконатися, що модуль може розпізнавати та повідомляти про проблеми, пов’язані з його основною функціональністю – моделлю машинного навчання, яка є важливою для його роботи.

Ретельне тестування цих сценаріїв гарантує, що модуль класифікації є не тільки функціональним, але й зручним для користувача та стійким до типових проблем, які можуть виникнути під час використання в реальному світі. Такий підхід до тестування зміцнює впевненість у здатності модуля безперебійно працювати в різних середовищах і обробляти помилки таким чином, щоб підтримувати належну взаємодію з користувачем.

Протестовано програмний модуль за різними сценаріями, щоб переконатися в стійкості та надійності; тести включали обробку відсутніх файлів, порожніх файлів і проблем із завантаженням моделі з відповідними повідомленнями про помилки, реалізованими для кожного сценарію. Ці тести підтвердили, що модуль надійно працює та забезпечує високу точність класифікації.

# ВИСНОВКИ

Під час виконання кваліфікаційної роботи було:

– проаналізовано способи представлення природної мови для комп'ютерних систем та підходи до обробки природної мови засобами штучного інтелекту;

– проведено аналіз існуючих рішень для класифікації елементів анотацій наукових статей на основі штучного інтелекту та сформовано задачу дослідження;

– представлено структуру програмного модуля класифікації елементів анотацій наукових статей на основі штучного інтелекту;

– розроблено архітектуру моделі обробки природної мови для виконання класифікації елементів анотацій наукових статей, взявши за основу трансформерну архітектуру BERT та використано додаткові види інформації як вхідні дані моделі; використання гібридної моделі дозволило покращити розуміння змісту і структури анотацій, підвищивши точність класифікації;

– представлено алгоритмічне забезпечення програмного модуля;

– здійснено навчання моделі обробки природної мови та перевірено точність класифікації; процес навчання передбачав попередню обробку даних, включаючи токенізацію, одноразове кодування числових вхідних даних і групування для ефективної обробки;

– реалізовано програмний модуль;

– для реалізації програмного модуля та навчання моделі було використано мову програмування Python, бібліотеку машинного навчання Tensorflow, середовище Visual Studio Code та середовище керування версіями бібліотек Anaconda;

– протестовано програмний модуль за різними сценаріями, що підтвердило його потенціал для автоматизації класифікації елементів анотацій наукових статей, що робить його цінним інструментом для науковців та дослідників. Завдяки категоризації речень модуль допомагає швидко переглядати відповідні дослідження, тим самим прискорюючи роботу науковцям.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Buenz E. J. Essential elements for high-impact scientific writing. Nature, 2019. https://www.nature.com/articles/d41586-019-00546-7

2. Plaxco K. W. The art of writing science. Protein Science, 2010. Pp. 2261-2266.

3. Babić K., Meštrović A. Survey of Neural Text Representation Models. Information, 11(11), 2020. p. 511. https://doi.org/10.3390/info11110511

4. Nadkarni P. M., Chapman, W. W. Natural language processing: An introduction. Journal of the American Medical Informatics Association, 18(5), 2011. pp. 544-551. https://doi.org/10.1136/amiajnl-2011-000464

5. Aranha R. V., Corrêa C. G., Nunes F. L. S., Adapting Software with Affective Computing: A Systematic Review. IEEE Transactions on Affective Computing, vol. 12, no. 4. 2021. pp. 883-899.

6. The marvelous mathematics of computational linguistics. URL: https://www.technologyreview.com/2015/09/17/166211/king-man-woman-queen-the-marvelous-mathematics-of-computational-linguistics/

7. Hochreiter S., Schmidhuber J. Long Short-term Memory. Neural computation. 1997.

8. Huang Z., Xu W., Yu K. Bidirectional LSTM-CRF Models for Sequence Tagging. ArXiv. 2015. /abs/1508.01991

9. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., Polosukhin I. Attention is all you need. 2017.

10. Devlin J., Chang M., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018.

11. Dernoncourt F., Lee J. Y. PubMed 200k RCT: a Dataset for Sequential Sentence Classification in Medical Abstracts. In Proceedings of the Eighth International Joint Conference on Natural Language Processing, 2017, Volume 2, pp. 308–313.

12. PubMed 200k RCT. URL: https://www.kaggle.com/datasets/matthewjansen/pubmed-200k-rtc/data

13. Autoclass. URL: https://huggingface.co/docs/transformers/model_doc/auto

14. Use a GPU. URL: https://www.tensorflow.org/guide/gpu

15. Bert. URL: https://huggingface.co/docs/transformers/en/model_doc/bert

16. Essential documentation to Tensorflow. URL: https://www.tensorflow.org/guide

17. Softmax function. URL: https://en.wikipedia.org/wiki/Softmax_function

18. Sokolova M., Japkowicz N., Szpakowicz S. Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. AI 2006: Advances in Artificial Intelligence, Lecture Notes in Computer Science. Vol. 4304. 2006.

19. Tensorflow model. URL: https://www.tensorflow.org/api_docs/python/tf/keras/Model

20. Xu Y, Goodacre R. On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning. 2018.

21. Python. URL: https://www.python.org/

22. Tensorflow. URL: https://www.tensorflow.org/

23. Visual Studio Code. URL: https://code.visualstudio.com/

24. Anaconda Navigator. URL: https://docs.anaconda.com/free/navigator/index.html

25. Шелюжак Я.С., Турченко І.В. Програмний модуль класифікації елементів анотацій наукових статей. Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення: Матеріали міжнародної науково-практичної інтернет-конференції, випуск 89, 12-13.06.2024 р. URL: http://www.konferenciaonline.org.ua/ua/article/id-1786/

26. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.