

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

“_____” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему

**“Програмне забезпечення системи ідентифікації авторства тексту на основі
штучних нейронних мереж”**

Виконав здобувач вищої освіти

IV курсу, групи КН-22

ОПП «Комп’ютерні науки»

спеціальності 122 «Комп’ютерні науки»

_____ Півненко О.М.

«_____» _____ 2026 р.

Керівник проєкту

кандидат технічних наук, доцент

_____ Дресєв О.М.

«_____» _____ 2026 р.

Рецензент _____

Центральноукраїнський національний технічний університет
Факультет Механіко-технологічний
Кафедра Кібербезпеки та програмного забезпечення
Рівень вищої освіти бакалавр
Галузь знань 12 “Інформаційні технології”
Спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна (освітньо-наукова) програма “Комп’ютерні науки”

ЗАТВЕРДЖУЮ

Завідувач кафедри

д.т.н., проф.

Олексій СМІРНОВ

року

**ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА
ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ**

Півненку Олександр Михайловичу

(прізвище, ім’я, по батькові)

1. Тема роботи Програмне забезпечення системи ідентифікації авторства тексту на основі штучних нейронних мереж

2. Керівник роботи Дресєв Олександр Миколайович, канд. техн. наук, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №116-02 від 06.02.2026 року

3. Строк подання студентом роботи до захисту 09.06.2026

4. Мета та завдання випускної кваліфікаційної роботи Метою роботи є розробка програмного забезпечення системи ідентифікації авторства тексту на основі штучних нейронних мереж

5. 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

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи 3 аркуша

Функціональна схема системи 3 аркуша

Діаграма процесів 1 аркуш

Блок-схема алгоритму роботи додатку 1 аркуш

Блок-схеми алгоритмів роботи підпрограм 2 аркуша

Математична модель системи 1 аркуш

7. Дата видачі завдання 06.02.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів курсового проекту за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2026	
2.	Постановка задачі, оформлення ТЗ	15.03.2026	
3.	Розробка моделі компонента	20.03.2026	
4.	Розробка структур даних	25.03.2026	
5.	Розробка алгоритмів зв'язку та відображення	30.03.2026	
6.	Програмування алгоритмів	10.04.2026	
7.	Оформлення ПЗ	17.04.2026	
8.	Попередній захист роботи	16.05.2026	

Дата видачі завдання
06.02.2026 р.

Підпис керівника

_____ Дресєв О.М.
(прізвище та ініціали)

Завдання прийнято до виконання

_____ р.

Підпис здобувача

_____ Півненко О.М.
(прізвище та ініціали)

АНОТАЦІЯ

Півненко О.М. Програмне забезпечення системи ідентифікації авторства тексту на основі штучних нейронних мереж. 122 «Комп'ютерні науки». Центральноукраїнський національний технічний університет. Кропивницький. 2026.

У даній дипломній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмну систему, яка призначена для автоматичної ідентифікації авторства текстових фрагментів, що написані українською мовою.

Метою розробки є створення програмного забезпечення системи ідентифікації авторства тексту на основі штучних нейронних мереж.

Результат роботи – це програмна реалізація системи ідентифікації авторства тексту базі сіамської LSTM-мережі з інтерфейсом командного рядка.

У процесі виконання проведено аналіз існуючих підходів до ідентифікації авторства, аргументовано вибір архітектури сіамської двонаправленої LSTM-мережі. Розроблено конвеєр збору та підготовки текстових даних із веб-джерел, модуль токенізації, який враховує морфологію української мови, механізм генерації пар текстових фрагментів для навчання та валідації, нейромережеву модель бінарної класифікації та інтерфейс командного рядка для подальшого інференсу (тобто порівняння текстів).

Система підтримує вхідні формати JSON, YAML та TXT, автоматичне розбиття довгих текстів на фрагменти та агрегацію результатів у випадку якщо фрагменти перевищують максимальну допустиму довжину. Результати виводяться у текстовому, JSON або XML форматі.

Програма розроблена на основі мови Python із використанням фреймворку PyTorch, Ollama.

Ключові слова: комп'ютерні науки, ідентифікація авторства, верифікація авторства, нейронні мережі, сіамська нейронна мережа, LSTM, обробка природної мови, машинне навчання

ABSTRACT

Pivnenko O.M. Software for a Text Authorship Identification System Based on Artificial Neural Networks. 122 «Computer Science». Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

In this diploma thesis for the first (bachelor's) level of higher education, a software system has been developed for the automatic identification of authorship of text fragments written in Ukrainian.

The purpose of the development is to create software for a text authorship identification system based on artificial neural networks.

The result of this work is a software implementation of a text authorship identification system based on a Siamese LSTM network with a command-line interface.

In the course of this work, an analysis of existing approaches to authorship identification was conducted, and the choice of a Siamese bidirectional LSTM network architecture was justified. The following components were developed: a pipeline for collecting and preparing textual data from web sources, a tokenization module that accounts for the morphology of the Ukrainian language, a mechanism for generating pairs of text fragments for training and validation, a neural network model for binary classification, and a command-line interface for subsequent inference (i.e., text comparison).

The system supports JSON, YAML, and TXT input formats, automatic splitting of long texts into fragments, and aggregation of results in cases where fragments exceed the maximum allowable length. Results are output in plain text, JSON, or XML format.

The software was developed using the Python programming language with the PyTorch framework and Ollama.

Keywords: computer science, authorship identification, authorship verification, neural networks, Siamese neural network, LSTM, natural language processing, machine learning

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ	2
ВСТУП.....	3
1.1 Призначення системи.....	6
1.2 Область застосування.....	7
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ	10
2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.....	10
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування.....	26
2.3 Розгорнута постановка завдання	35
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ	36
3.1 Опис функціонування системи	36
3.2 Розробка структурної схеми.....	52
3.3 Розробка функціональної схеми	57
3.4 Розробка діаграми процесів.....	62
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ.....	65
4.1 Блок-схеми та опис алгоритмів функціонування системи.....	65
4.2 Захист розробленого програмного забезпечення.....	75
5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ	78
6 ОСНОВНІ ВИСНОВКИ.....	83
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	86

					ВКРБ-122.26.0025.00.00.ПЗ				
Вим	Арж.	№ докум.	Підп.	Дата					
Розроб.	Півненко О.М.				Програмне забезпечення системи ідентифікації авторства тексту на основі штучних нейронних мереж		Літ.	Аркуш	Аркушів
Перев.	Дресєв О.М.						Б	1	90
							КН-22		
Н.контр.	Коваленко А.С.								
Затв.	Смірнов О.А.								

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

LLM – Велика мовна модель (Large Language Model)

ПЗ – Програмне забезпечення (Software)

АЗ – Апаратне забезпечення (Hardware)

IDE – Інтегроване середовище розробки (Integrated Development Environment)

ML – Машинне навчання (Machine Learning)

ПК – Персональний комп'ютер

CPU – Центральний процесор (Central Processing Unit)

GPU – Графічний процесор (Graphical Processing Unit)

Embedding – вкладення (векторизоване представлення даних у машинному навчанні)

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

2

ВСТУП

На даний момент світ швидко змінюється під впливом інформаційних технологій. У повсякденне життя людей стрімко інтегруються цифрові сервіси для взаємодії із державою, із приватними закладами, з іншими людьми. Побут людей, їх звички та правила життя стрімко цифровізуються. Водночас, у кіберпростір переносяться не тільки послуги та можливості, але ще й злочинність. На даний момент у мережі Інтернет існує багато інформації, яку потрібно з тої чи іншої причини ідентифікувати (тобто визначити авторство тексту). Це може бути обумовлено захистом інтелектуальних прав, діяльністю правоохоронних органів, усуненням спамерів чи ботів на онлайн-платформах тощо. Проте, слід зауважити, що це може мати багато етичних, юридичних та інших колізій. Це обумовлено тим, що системи ідентифікації авторства тексту можуть бути використані різними суб'єктами: це можуть бути злочинні організації, державні структури або фізичні особи. Також слід зауважити, що навіть у випадку застосування таких систем відповідними органами, які мають на це право, постає питання правомірності їх використання.

Загалом питання ідентифікації авторства тексту було актуальним протягом усього існування людства. Багато королів, монархів, полководців, намагалися дізнатися більше про своїх опонентів, політичних супротивників та інших. Жандармерія минулих часів (і сучасна) намагається дізнатися більше про анонімних письменників, авторів творів та поетів, аналізуючи їх книги, поезію та їх літературний стиль. Яскравим прикладом є процес пошуку терориста під псевдонімом «UnaBomber» [1]. Цією людиною виявився Теодор Качінський – відомий математик, еколог та анархіст. І цьому пошуку сприяв аналіз його творів та маніфестів. Його не могли знайти за допомогою відбитків пальців або біологічних експертиз, тому прийшлося використати інші методи. Лише завдяки тому що ФБР погодило із журналістами публікацію твору

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

3

Качинського «Індустріальне суспільство та його майбутнє», його змогли затримати (публікація твору була також ультиматумом від Теодора, щоб запобігти майбутнім терактам). Але слід зауважити, що це була випадковість. Ідентифікувати Качинського змогли лише завдяки його рідним та близьким, які знали характерний стиль його письма (і основні орфографічні, лексичні помилки, що були унікальними лише для нього). Тобто це була «Інтуїтивна» система авторства тексту.

Із появою Інтернету ідентифікація авторства тексту стала ще більш актуальною. На даний момент існують багато корпусів текстів, які потрібно проаналізувати та деанонімізувати їх авторів. Наразі це потрібно робити в автоматизованому режимі.

Метою роботи є створення системи ідентифікації авторства тексту. Це повинно бути програмне забезпечення, яке працює на базі операційної системи (дистрибутивів) Linux або ж Windows. Воно повинно використовувати GPU для більш ефективного обчислення ідентифікації тексту. Проте потрібно зберігати можливість обчислення на CPU (якщо у користувача не буде графічного процесору). Це повинно бути програмне забезпечення, що порівнює два екземпляри тексту і видає ймовірність їх співпадіння (тобто ймовірність того що їх написала одна людина).

Для досягнення поставленої мети повинен бути план реалізації програмного забезпечення (який також буде включати супутню інформацію щодо питань, пов'язаних із контейнеризацією, операційними системами та адаптацією ПЗ під апаратні можливості):

- Дослідження PyTorch Framework, JupyterNotebook, Ollama;
- Аналіз існуючих рішень, архітектур та способів реалізації данного ПЗ. Визначення майбутньої архітектури поміж різними моделями нейронних мереж: FNN (перцептрон), CNN (ResNet, VGG, EfficientNet, U-Net), RNN (LSTM, GRU, Bidirectional RNN), Transformer (BERT, GPT, ViT, T5), GNN (GCN, GAT, GraphSAGE);

- Створення нейронної мережі, підготовка датасету та реалізація додаткових функцій; розгортання локальних нейронних мереж на базі Ollama;
- Проведення парсингу даних із мережі Інтернет, перевірка отриманих даних, їх очистка (у тому числі за допомогою локальних моделей, що будуть допомагати створювати датасет для навчання);
- Навчання нейронної мережі, її тестування та відлагодження.

Практична цінність отриманих результатів полягає у тому, що створену нейронну мережу можна використати для допомоги правоохоронним органам у їх діяльності, для перевірки на плагіат, для використання у системі освіти України. Також це можна використати у соціальних мережах для більшої ефективності моніторингу користувачів, запобіганню цькуванню та порушенням правил платформ. Цілком органічно із вищенаведеного випливає те, що це також можна використати в індустрії розваг. Це стосується відеоігор, оффлайн-активностей тощо.

Важливо враховувати, що це програмне забезпечення може мати етичні колізії. Його можна використовувати як для запобігання цькуванню, так і для проведення. Також слід зауважити, що наприклад, правоохоронні органи в державах, які мають слабкі інститути, можуть використовувати це програмне забезпечення і подібні йому системи для пригнічення опонентів, політичних дисидентів та для ліквідації політичних супротивників. Оскільки політична діяльність нерозривно пов'язана із публікацією творів, маніфестів та інших літературних надбань людини.

Загалом, питання етичності лягає на ту людину або організацію, що її використовує. Також це можна використовувати для захисту від вищенаведених загроз. Наприклад, дисидент може перевірити свої твори, чи не співпадають вони, і якось їх перефразувати.

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

5

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Слід розділити систему на дві складові: програмне забезпечення і натреновану нейронну мережу. Програмне забезпечення буде являти собою вихідний код на мові програмування Python. Це будуть файли формату PY та/або IPYNB. Водночас, натренована нейронна мережа буде мати формат PTH [2]. Система буде призначена для розпізнавання двох фрагментів тексту і видачі ймовірності, наскільки вони схожі. Система буде призначена винятково для українських текстів. Це обумовлено тим, що, по-перше, вже існують системи та програмне забезпечення, призначені для розпізнавання тексту англійською мовою. Водночас систем, призначених для розпізнавання української, немає. Розмір вхідних фрагментів буде залежати від архітектури нейронної мережі.

Важливим елементом характеристики буде той факт, що потрібно визначити апаратне забезпечення, необхідне для функціонування системи. Слід окремо виділити апаратне забезпечення, що потрібно для запуску вже натренованої нейронної мережі, а також апаратне забезпечення, яке потрібно для тренування нейронної мережі. Безумовно, для тренування/створення моделі потрібно будуть значно більші обчислювальні ресурси.

Важливим елементом системи є супутнє програмне забезпечення, тобто залежності (dependencies). До цього списку можна віднести: CUDA, Python Interpreter (3.12 і вище версії), PyTorch, Ollama. Вони є необхідною умовою, оскільки для роботи і тренування нейронної мережі потрібно взаємодіяти із апаратним забезпеченням. Для інших платформ, скажімо, графічних процесорів AMD, можуть існувати альтернативи. Графічні процесори AMD мають технологію ROCm [3], яка є альтернативою

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

6

пропрієтарній CUDA від компанії NVIDIA.

Призначення системи є розпізнавання фрагментів тексту. Це можна робити в залежності від потреб фізичних чи юридичних осіб. Програмний комплекс не передбачає використання графічного інтерфейсу користувача. Для використання даної системи потрібно буде взаємодіяти із інструментами розробника, терміналом.

1.2 Область застосування

Система може мати застосування у багатьох предметних областях. Окрім цього, її можна також розвинути, доробити, провести рефакторинг і покращити її функціональність. Можна збільшити кількість даних для тренування, можна збільшити точність і покращити алгоритми, змінити підхід до створення embeddings. Можна також інтегрувати у програмну систему графічний інтерфейс користувача.

Але знову ж слід наголосити, що дана система може мати певні етичні та юридичні колізії. Тому потрібно зважати як на тих, хто може використовувати цей програмний комплекс, так і на тих, відносно (проти) кого застосовується це програмне забезпечення.

Програмне забезпечення для ідентифікації авторства тексту може знадобитись у наступних областях людської діяльності:

- Криміналістика. Програмне забезпечення можна використовувати для ідентифікації авторства тексту, який може належати злочинцю. Це може бути додатковим аргументом під час судових засідань, проведення слідчих дій та загалом при оперативній діяльності правоохоронних органів;
- Історія. Спеціалісти, що займаються історією людства, можуть використати це програмне забезпечення для дослідження літературних творів. Частина творів у минулому та й зараз пишеться анонімно. Для того, щоб встановити авторство, можна використати це ПЗ. Завдяки цьому ми можемо

Вим	Арк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

7

більше дізнатись про літературні твори минулих часів, про анонімних авторів та письменників, що намагались приховати свою ідентичність;

- Антиплагіат. Програмне забезпечення можна використати задля ідентифікації тексту, який студент міг взяти із інших робіт. Також можна використати це у тому випадку, якщо є підозра на використання штучного інтелекту. Також це можна використати у інших питаннях, якщо є підозра на порушення академічної доброчесності;

- Корпоративна безпека. ПЗ можна використати для того, щоб дізнатися про можливі витіки інформації із компанії. За допомогою цього інструменту можна відстежити недоброчесних співробітників, які передають інформацію про внутрішні процеси компанії конкурентам. Це відноситься як до фінансово-економічних показників, так і до технологічних даних, креслень, тощо;

- Веб-комерція. Програмне забезпечення для порівняння текстів та ідентифікації авторства можна використовувати у сфері комерції. До цього можна віднести соціальні мережі, маркетинг, пошук ботів тощо. За допомогою цього програмного комплексу можна відшукувати спамерів, ботів, запобігати маніпуляції суспільної думки. Окрім вищенаведеного, можна також відшукувати тих, хто порушив правила платформи (і зареєстрував новий акаунт, скажімо).

Загалом, це лише частина сфер, де можна застосувати це програмне забезпечення. Але, безумовно, слід зауважити, що у цього ПЗ є етичні проблеми. Ризики можуть бути найрізноманітніші. До них можна віднести, скажімо, цитування користувачів. Навіть якщо вони зареєструють новий акаунт, то за допомогою цього ПЗ можна знайти їх за їх характерним стилем письма. Можуть бути і інші проблеми. Наприклад, коли будуть застосовувати це програмне забезпечення і блокувати людей превентивно, хоча вони не будуть спамерами, ботами, і просто будуть «видавлювати» їх через політичну позицію. А це також може мати ризики для інтернет-цензури. Маючи такий

Вим	Анк	№ докум.	Підпис	Дата

сильний інструмент, спеціальні служби можуть використати його для маніпуляції суспільною думкою. Можна за допомогою цього інструменту видаляти публікації політичних дисидентів та тих людей, які можуть бути нелояльними до політичного режиму. На даний момент вже існують приклади такого підходу: коли письменників, публічних діячів та правозахисників видаляють із публічного простору. До країн, де така практика має місце, можна віднести Китайську Народну Республіку, Еритрею, Саудівську Аравію, Ісламську Республіку Іран, Кубу, Алжир, Південно-Африканську Республіку, Анголу, Нікарагуа, Сальвадор тощо [4].

Важливим фактором тут є той факт, що з появою такого інструменту в руках людей їм можуть все одно скористуватись навіть у демократичних країнах. Це можна побачити на прикладі ситуації із Едвардом Сноуденом, який розкрив програми збору даних США [5]. Вони навіть шпигували за своїми безпосередніми союзниками по НАТО: збирали дані, читали дипломатичні переписки, і аналізували увесь трафік, що проходив через сервери американських компаній [6] (із ними, безумовно, домовилось Агентство Національної Безпеки). Якби у «чекистів» була можливість та обчислювальні ресурси порівнювати тексти, і була нагальна необхідність у цьому, то вони робили б це, маючи усі дані на руках.

Навіть у найбільш демократичних країнах правоохоронні органи будуть зловживати таким видом програмного забезпечення і використовуватимуть його із метою комусь зашкодити. Тому таке програмне забезпечення може мати подвійне призначення. Його можуть використовувати як в етично прийнятних цілях, так і в цілях неприйнятних, які можуть комусь зашкодити. Повна відповідальність лягає на тих, хто використовуватиме це програмне забезпечення та буде проводити діяльність.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

На даний момент вже існують системи технології, архітектури, програмні рішення, які дозволяють проводити ідентифікацію авторства тексту. Не всі з них підтримують можливість ідентифікації авторства саме українською мовою, оскільки багато з них призначені для англomовних користувачів. Ці системи можуть розроблятися як з боку приватних підприємств (акціонерні товариства, товариства із обмеженою відповідальністю), які мають на мету отримання прибутку, так й від академічної спільноти, а також і від ентузіастів. Вони мають дуже велику кількість ліцензій. Серед них можуть бути, наприклад, пропріетарні ліцензії, ліцензії на відкрите ПЗ тощо. Існують варіанти запуску в хмарному середовищі, а також локально на комп'ютері дослідника.

Не всі програмні комплекси використовують засоби машинного навчання (нейронні мережі) для розпізнавання текстів. Частина програмного забезпечення може використовувати метод опорних векторів, дерева рішень або байєсові класифікатори. Також можуть використовувати статистичні методи, які базуються не на нейронних мережах, а на методах лінгвістичному аналізу тощо.

Першим варіантом, який можна розглянути, буде програма JGAAP (Java Graphical Authorship Attribution Program) – це відкрите програмне забезпечення, яке ліцензується під AGPL V3 та дозволяє визначити авторство тексту. Вона представляє графічний інтерфейс користувача, а також різну кількість методів, які дозволяють проводити ідентифікацію. Програмне

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

10

забезпечення написано на мові програмування Java. Приклад використання програмного забезпечення можна побачити на рисунку 2.1. Там зображено головне вікно програми, де можна побачити графічний інтерфейс. Програма зараз знаходиться у відкритому доступі на платформі GitHub, де кожна людина може долучитись до розробки.

Серед головних переваг можна виділити:

- Відкритість вихідного коду. Це дозволяє провести незалежний аудит, провести самостійну компіляцію та збірку виконуваного файлу. Також можна провести модернізацію або ж модифікацією програмного забезпечення задля зміни або ж отримання необхідного функціоналу. Якщо вихідний код можна змінювати, то це позитивно впливає на кількість плагінів та супутнього ПЗ, що використовується у цьому комплексі;
- Безкоштовність. Програмне забезпечення JGAAP можна використовувати повністю у комерційних цілях і не платити за це гроші;
- Реалізація більшості простих методів ідентифікації. Якщо поставлена задача не потребує екзотичних методів визначення авторства тексту, то прості методи, як-от, наприклад: дерева рішень, метод опорних векторів, байєсові класифікатори, дозволяють ефективно проводити розмежування фрагментів тексту.

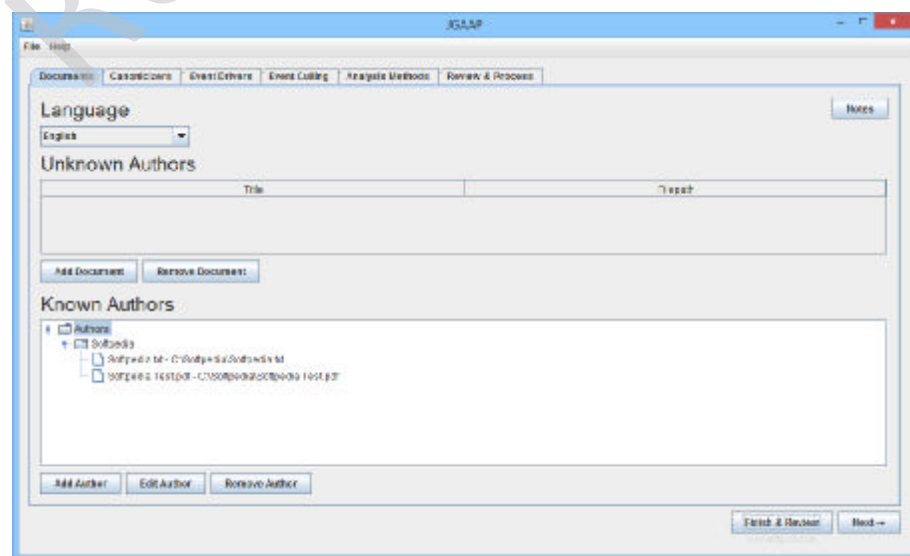


Рисунок 2.1 – Головне вікно програми JGAAP [7]

Серед негативних факторів можна виділити:

- Неможливість приватно модифікувати програмне забезпечення. Оскільки програмне забезпечення розповсюджується під ліцензією AGPL V3, то виходить така ситуація, що будь-яка зміна вихідного коду з боку, наприклад, приватної компанії, повинна стати публічною. Тобто кожна людина зможе перевірити та передивитись ці зміни. Якщо ж компанія, тобто юридична особа або фізична особа будуть приховувати зміни в вихідному коді, то це може призвести до проблем, а саме судової тяганини;
- Застарілі програмні рішення. Програмне забезпечення написано на мові програмування Java. Це є дуже великим мінусом, оскільки зараз для технологій машинного навчання ця мова не використовується. [8] Також в програмі реалізовані застарілі методи ідентифікації, які зараз поступилися нейронним мережам. Тобто точність вимірювань може бути низькою. Додатково, слід зауважити, що у програмі, оскільки вона має відкритий вихідний код, немає хмарних обчислень, що потребує більших ресурсів комп'ютера користувача. Це призводить до того, щоб використати це програмне забезпечення потрібно витратити набагато більше часу, ніж при використанні підходів із застосуванням альтернатив у вигляді хмарних обчислень;
- Пауза проекту. На даний момент, якщо подивитись репозиторій на Github, а також офіційний сайт, то можна зрозуміти, що програмне забезпечення майже не змінювалось. А на даний момент офіційний сайт не відкривається у браузері. Це говорить про те, що баги, недоліки у програмі будуть усунуті, можливо, нескоро.

Загалом, система JGAAP на момент створення виглядала дуже перспективною. Але оскільки зараз розвиток машинного навчання базується на інших мовах програмування, використовує інші фреймворки, програмні рішення та підходи, зараз ця програма застаріла. Проте, якщо потрібні прості методи ідентифікації, або ж, скажімо, потрібно модифікувати вже існуючий

функціонал, при цьому знання програмістів обмежуються інфраструктурою навколо мови Java, то можна вибрати саме це програмне забезпечення.

Програмне забезпечення JGAAP для обчислень використовує CPU. це призводить до того, що у разі надмірних обчислень програмне забезпечення може просто «покласти» систему. У разі якщо потрібно відмінити обчислення, такої можливості немає. Потрібно чекати, допоки вони виконуються, або використовувати системні монітори «Диспетчер задач» або «GNOME System Monitor» для зупинки. Це зображено на рисунку 2.2, 2.3 та 2.4. Також це може призводити до візуальних артефактів, які негативно впливають на сприйняття даних. Графічний інтерфейс користувача існує (тобто це не утиліта командного рядку), але він не є інтуїтивно зрозумілим, має проблеми із відгуком на дії користувача, занадто малий шрифт.

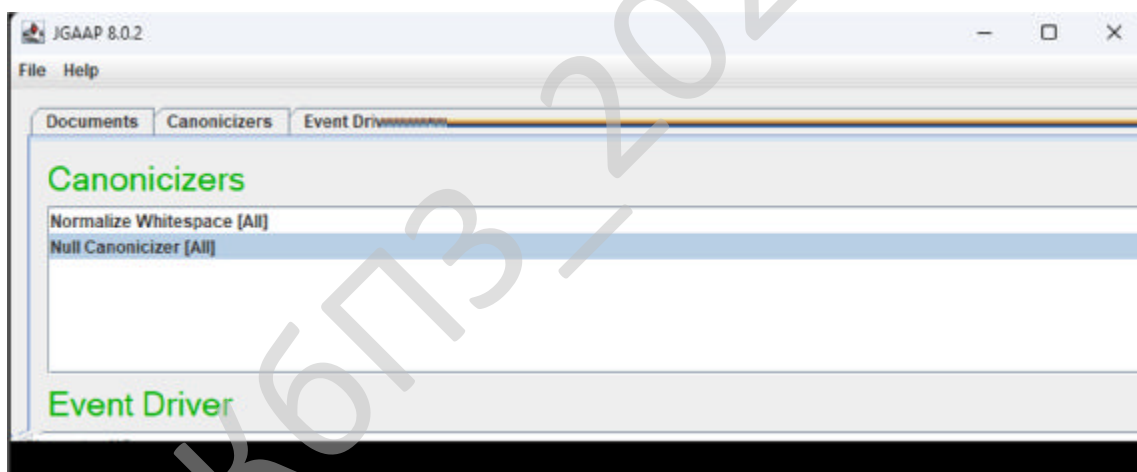


Рисунок 2.2 – Нemoжливiсть зупинити обчислення (разом із графічними артефактами)

Name	Status	CPU	Memory	Disk	Network
OpenIDK Platform binary		82.3%	8,396.3 MB	0.1 MB/s	0 Mbps
Task Manager		1.0%	66.9 MB	0.1 MB/s	0 Mbps
Desktop Window Manager		0.4%	51.1 MB	0 MB/s	0 Mbps

Рисунок 2.3 – Виділення великої кількості ресурсів процесу JGAAP та можливий збій системи у випадку нестачі ОЗП

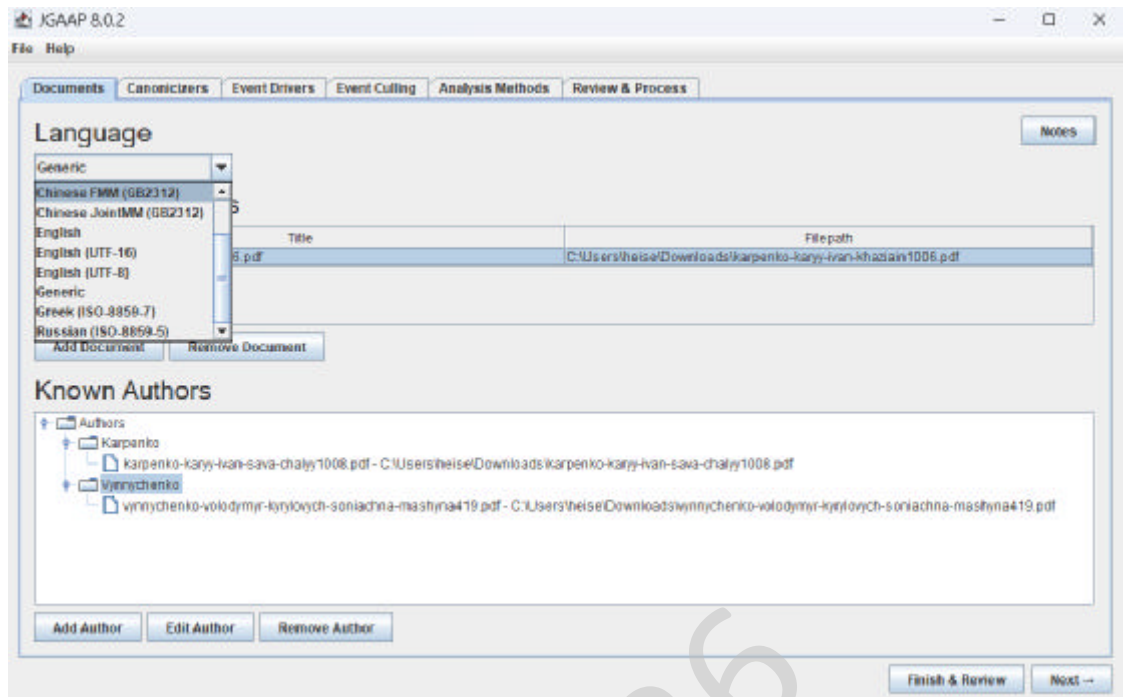


Рисунок 2.4 – Головне вікно програми із вибором мов, які підтримує ПЗ, а також твори, які використовувались для тестування

Напевно, найбільшою проблемою ПЗ JGAAP є той факт, що у ньому реалізовано closed-set розпізнавання, а також відсутність української мови. Фактично, це робить програму такою, що її не можна використовувати для українських текстів. Більш перспективним є використання цієї програми, якщо є текст китайською, англійською, грецькою чи арабською мовою. Особливо це важливо в тому випадку, якщо користувач, що буде використовувати програму, не має достатньої кваліфікації для глибоких налаштувань.

Closed-set розпізнавання є досить серйозним недоліком. Такий підхід призводить до того, що програма завжди «видає» конкретного автора. Вона не може сказати, що жоден з авторів не співпадає і взагалі не відомо, хто це міг написати. Безумовно, можливо, за допомогою комбінації методів, що реалізовані у ПЗ, можна досягти більш точної інформації для вимірювання розпізнавання тексту, але це не є повноцінна сучасна система.

Напевно, найголовнішою проблемою можна також визначити

відсутність засобів ідентифікації за допомогою нейронних мереж. Старі методи машинного навчання, які використовували більш прості та емпіричні алгоритми, хоча і є інтуїтивно зрозумілими, а також потребують меншої кваліфікації з боку програміста чи математиків, що адаптовували алгоритми під конкретні задачі, але наразі показують значно гірші результати, ніж використання більш сучасних підходів штучного інтелекту. Водночас, вони потребують менше обчислювальних ресурсів для навчання, тому для локального запуску це було виправдано.

Наступним програмним забезпеченням, яке слід розглянути, це NeoNeuro. Його створила компанія, що базується на Кіпрі [9]. Компанія розробляє велику кількість продуктів, але під час дослідження важлива саме програма для ідентифікації авторства тексту, а саме Authorship Attribution Software.

На даний момент програмне забезпечення не розвивається, але його сторінка все ще присутня на сайті компанії. Слід зауважити, що оскільки програма пропрієтарна і будь-які способи дізнатись про внутрішню архітектуру та програмні рішення натикаються на те, що потрібно проводити реверс-інженерінг. Тому оцінити ефективність даного програмного забезпечення не представляється можливим. Безкоштовна версія, яка пропонується компанією, може містити загрози (бекдори), які потім будуть слугувати для кібератак. Варто у такому випадку проводити запуск програмного забезпечення на віртуальній машині або в іншому контейнеризованому середовищі. Але є дуже важливе зауваження: при спробі завантажити архів, а потім провести його деархівацію, виникла проблема – архів був пошкоджений. У такому випадку під час аналізу ПЗ слід спиратись саме на наукові статті, публікації та іншу інформацію, яку можна знайти в публічних джерелах щодо цього ПЗ. Окрім технічного питання, також будуть обговорені етичні аспекти використання.

У публічному полі не було знайдено згадки про це програмне

Вим	Анк	№ докум.	Підпис	Дата

забезпечення в наукових базах даних Scopus, WebScience та Google Scholar. Це свідчення того, що програмне забезпечення не змогло стати повноцінним гравцем на ринку систем ідентифікації авторства тексту.

Програмне забезпечення відсутнє у відкритих базах даних. Наразі складно стверджувати, чи дійсно воно використовується певними державними структурами (хоча, без сумніву, може застосовуватися інше ПЗ з аналогічним функціоналом). Технічна можливість відкривати лише частину реєстрів закупівель та приховувати сам факт інтеграції таких програм у кінцевих користувачів цілком існує. Головне вікно системи можна побачити на скріншоті, який опублікувала сама компанія NeoNeuro (зображено на рисунку 2.5).

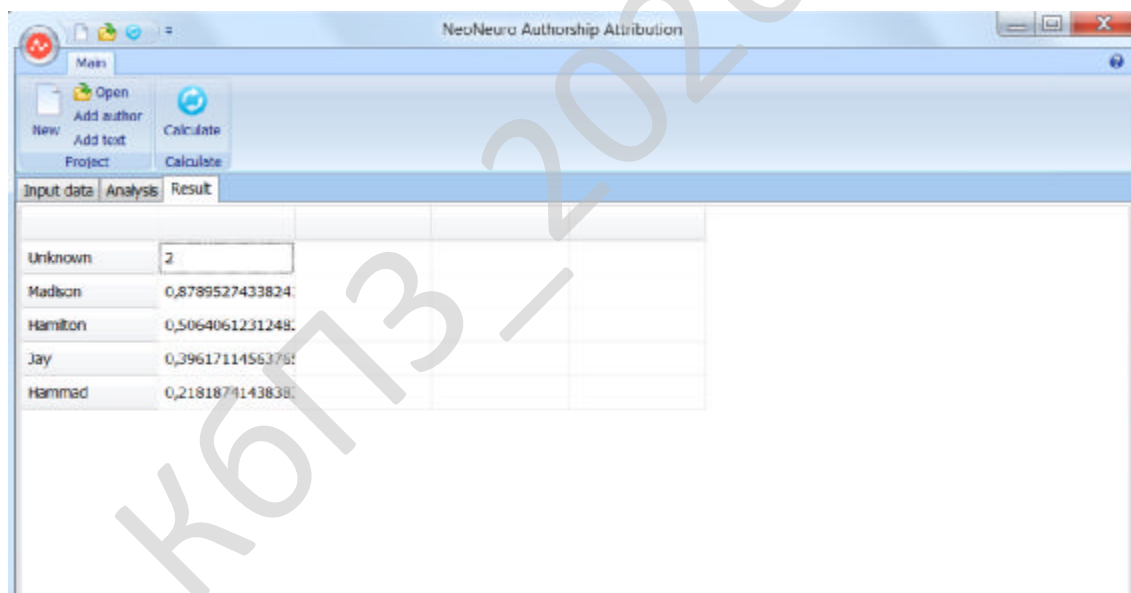


Рисунок 2.5 – Зображення головного вікна Authorship Attribution [10]

Якщо придивитись, то можна побачити, що головне вікно програми має інтерфейс, схожий на стандартний, для вікон в операційній системі Windows 7. Про це каже хрест у правому верхньому куті, а також прозорість вікна (що було головним елементом саме у цій версії Windows). Враховуючи, що Windows 7 перестала підтримуватись із 2018 року, а також, що саме у цей рік Windows 10 набула більшої популярності [11], аніж минула версія Windows,

то програмне забезпечення вже давно не оновлювалось (інакше б оновили скріншоти на сайті). Динаміка зміни популярності зображена на рисунку 2.6.

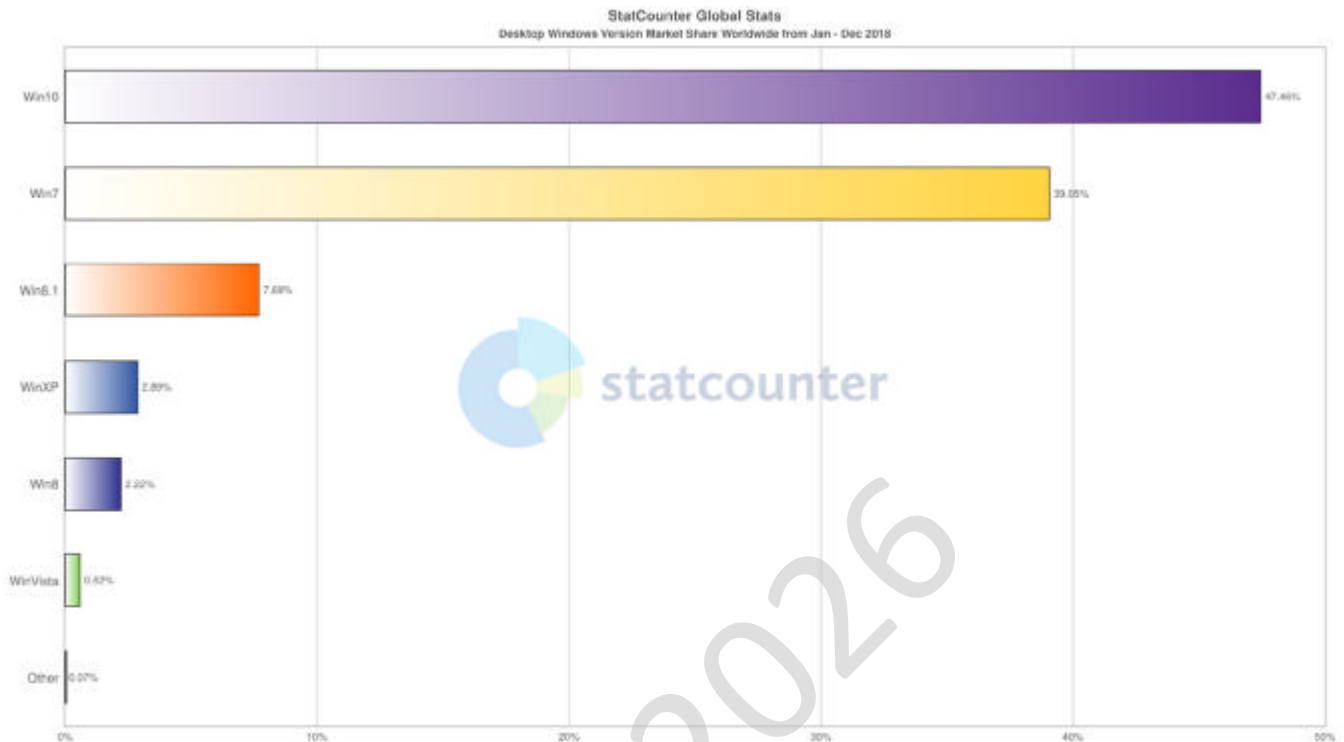


Рисунок 2.6 – Динаміка зміни популярності версій Windows

Оцінити функціонал через закритий вихідний код, а також неможливість декодувати архів із програмою, не представляється можливим. Але судячи із скріншотів, а також опису функціоналу програми, присутня та ж сама проблема, що із JGAAP: вона реалізовує визначення авторства тексту через closet-set підхід. Тут проявляються усі ті ж самі недоліки, що і були вище наведені.

Значною проблемою також є етичний та безпековий аспекти використання програми NeoNeuro. З огляду на актуальні виклики та загрози безпеці, впровадження подібного софту є неприпустимим. Адаптувати власні системи чи аналізувати дані за допомогою цього продукту – це фактично добровільне створення критичної вразливості. Оскільки це програмне забезпечення має неблагонадійне походження, існує висока ймовірність

наявності в ньому бекдорів, програмних закладок чи інших прихованих механізмів, які можуть спростити проведення атак на критичну кіберінфраструктуру. Застосування такого інструменту державними та правоохоронними органами було б абсолютно безвідповідальним і шкідливим кроком. До того ж, ця програма не здобула популярності ні на міжнародному рівні, ні навіть на своєму первинному локальному ринку.

Водночас, якщо брати навіть інші країни, де правоохоронним органам може знадобитись це програмне забезпечення і вони не знаходяться у стані збройного конфлікту, то постає питання захисту даних користувачів, постає питання національної безпеки, а також той факт, що дане ПЗ буде експортуватись до країн (і експортувалось, певно, вже) із погано розвиненими інститутами держави. Це може створити базу для зловживань з боку чиновників у тих регіонах. Тобто, якщо open-source альтернативи можуть використовувати навіть фізичні особи і завдяки цьому, наприклад, якось протидіяти загрозам та розуміти можливі шляхи атаки на власну приватність, то у випадку пропрієтарного ПЗ ситуація стає гіршою (механізми реалізації алгоритмів, програмні рішення стають важко досліджуваними, і у такому випадку не представляється можливим прогнозувати, як буде діяти програмне забезпечення і як воно буде аналізувати інформацію). Слід також зауважити, що навіть використання суб'єктами цього ПЗ може призводити до проблем, а саме: програмне забезпечення вже давно не оновлюється, тоді виникає проблема, коли через, наприклад, баги у коді, можливі витоки пам'яті буде відбуватись кібератака на тих, хто застосовує цей інструмент. Тобто навіть сам факт застосування може призвести до підвищеної небезпеки у власній кіберінфраструктурі.

Серед українського сегменту можна знайти небагато програм у цій сфері. Однією з тих, що була знайдена під час дослідження даних систем, є програма TextAttributor [12]. Узагалі, це не зовсім є програмне забезпечення у тому розумінні, як про це думають багато користувачів. Дана система

ідентифікації авторства тексту є веб-додатком, де потрібно проводити реєстрацію, опісля аутентифікацію і подальше користування. Вебсайт не підтримує використання захищеного з'єднання за допомогою HTTPS. Це вже є дуже важливим недоліком, але про це далі буде йти мова нижче. Процедура реєстрації зображена на рисунках 2.7 та 2.8. Спочатку буде зображено вікно реєстрації, де потрібно увести свої дані, а на наступному рисунку буде вже показано кінцевий результат. Слід зауважити, що лист для підтвердження електронної пошти був отриманий не відразу. Для цього знадобилось декілька годин. Подібна ситуація не пов'язана із доменом електронної пошти, оскільки вона не міститься у чорних списках та є популярною серед української аудиторії.

Рисунок 2.7 – Вікно реєстрації

Рисунок 2.8 – Процес закінчення реєстрації

Вим	Анк	№ докум.	Підпис	Дата

Для додавання тексту використовується веб-інтерфейс і текст у ньому потрібно вводити вручну. Тобто немає можливості використовувати сучасні формати даних, як-от, скажімо, PDF, DOCX, ODF тощо. Це унеможливило будь-яке використання цього ПЗ у реальних задачах. Ніхто не буде копіювати десятки мегабайт тексту, а якщо і прийдеться (за відсутності альтернативи), то буде важко використовувати той інтерфейс, що надається розробником. Вікно додавання тексту зображено на рисунку 2.9. Приклад доданих текстів на рисунку 2.10

Рисунок 2.9 – Вікно додавання тексту

Назва тексту	Мітка тексту	Статус	Речень	Слів	
Сонячна машина уривок_0	В.Винниченко	В обробці	122.00	849.00	
Сонячна машина уривок_1	В.Винниченко	В обробці			

Рисунок 2.10 – Приклад доданих текстів (в обробці)

Демонстраційні приклади були створені таким чином, що були взяті уривки із роману Володимира Винниченка «Сонячна машина». Враховуючи кількість слів та розмір уривків, веб-додаток обробляє їх занадто довго.

Подібний розмір уривків іншими програмами оброблявся значно швидше, і також можна було визначати, які саме параметри обраховувати на базі цього тексту. В українському ж додатку обраховуються усі параметри, незалежно від того чи потрібні вони кінцевому користувачу, чи ні. Також було з'ясовано, що тексти обробляються послідовно, тобто в однопоточному режимі. Це є серйозним недоліком та проблемою системи. Приклад доданих текстів, один з яких вже оброблений, знаходиться на рисунку 2.11.

Мої тексти

[Додати новий текст](#)

Назва тексту	Мітка тексту	Статус	Речень	Слів	
Сонячна машина уривок_0	В.Винниченко	Оброблено	122.00	849.00	
Сонячна машина уривок_1	В.Винниченко	В обробці	304.00	1,846.00	

Рисунок 2.11 – Приклад доданих текстів (один оброблений)

Звертає на себе увагу той факт, що під час обробки текстів інший функціонал додатку або ж не працює, або ж видає помилку.

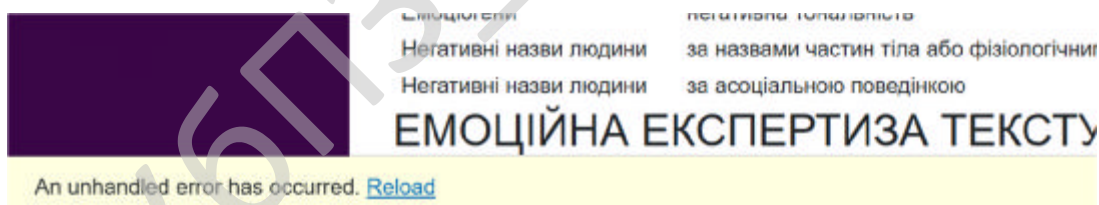


Рисунок 2.12 – Помилка під час перегляду індексів атрибуції тексту «Сонячна машина уривок_0»

Це може бути спричинено тим, що програмне забезпечення не пройшло належного тестування.

Оброблення двох невеликих уривків тексту зайняло більше ніж 20 хвилин. Єдиною перевагою, що можна відмітити у цьому випадку, це те, що сайт використовує обробку на боці серверу. Проте чи вартий такий підхід використання у даному випадку, залишається, як мінімум, дискусійним.

Слід також зауважити, що ця система має таку ж саму проблему, що і минулі різновиди програмного забезпечення, а саме: вона реалізовує closed-set підхід до розпізнавання авторів. Серед тих авторів, які використовуються для висновку схожості тексту, є лише О. Гончаренко, І. Фаріон, М. Безугла [13]. Можливо, це не всі автори, але ті, яких видала система. Приклад висновку зображений на рисунку 2.13.

ВИСНОВКИ НЕЙРОННОЇ МЕРЕЖІ ⁱ:

Ступінь токсичності тексту ⁱ: **0.50**

Ступінь подібності з текстами зазначених авторів ⁱ:

І. Фаріон - 0.61
М. Безугла - 0.24
О. Гончаренко - 0.15

Мітка групи тексту (автор/джерело/набір):

В.Винниченко

Рисунок 2.13 – Приклад висновку нейронної мережі

При спробі натиснути кнопку «Зберегти» ніяких дій не відбувається. Або про це не повідомляється користувачу за допомогою елементів інтерфейсу, або даний функціонал не є реалізованим у даній версії ПЗ. Дізнатись результат немає можливості.

Загалом, програмне забезпечення TextAttributor відрізняється наступними недоліками:

- Програмне забезпечення виконується в однопоточному режимі. Паралельне обчислення декількох текстів не представляється можливим. У користувача також немає можливості вибрати, які тексти потрібно опрацьовувати першими. Тобто, якщо було додано тексти в іншому порядку, то змінити порядок їх аналізу не представляється можливим. Також зупинити процес теж немає змоги;
- Застарілий графічний інтерфейс користувача. Хоча додаток і

знаходиться у вигляді веб-версії, але елементи взаємодії з користувачем представлені без застосування сучасного дизайну. Використовують стандартний набір HTML-тегів, мають скудний набір CSS-стилів. Загалом, інтерфейс представляє собою біле полотно із кнопками, текстом на них, картинками доволі низької якості. Веб-сайт не дає можливість зрозуміти користувачу, чи відреагував він на натискання елементів інтерфейсу. Частина кнопок та функціоналу мені не вдалося верифікувати: чи працюють вони насправді. Якщо у програмі виникає помилка, то користувачу просто показується повідомлення про це. Незрозуміло, як можна це виправити, сповістити розробників про баги на сайті, або ж які альтернативи варто спробувати;

- Погано реалізований процес реєстрації. Під час створення аккаунту лист не надходив на електронну пошту протягом багатьох годин. Тут варто звернути увагу на те, що домен пошти та, в принципі, електронний адрес не знаходяться у чорних списках. Його використовують багато україномовних користувачів у мережі інтернет, а саме «meta.ua». Інші відвідувачі сайту для своїх практичних потреб не чекали б декілька годин для підтвердження власного аккаунту. Тим паче, що є локальна альтернатива цьому ПЗ, згадана раніше, а саме JGAAP. Якщо ж люди будуть використовувати менш відомі домени, то це може спричинити проблеми або повну неможливість для реєстрації;

- Нереалізований процес визначення авторства тексту повноцінно. Незважаючи на те, що параметри, які обчислюються системою, може побачити користувач і зробити певні висновки, все ж це не дає однозначної відповіді, чи співпадають два уривки тексту. Де-факто використовувати даний інструмент для визначення авторству не представляється можливим. Статистичні методи не дають повної гарантії виконання задачі, водночас обчислюються довго, а нейромережевий підхід реалізований за принципом closed-set, що ставить крапку на його потенційному використанні у реальних ситуаціях (оскільки

співпадіння їх тексту із уривками Ірини Фаріон для більшості юридичних чи фізичних осіб критично не цікаво). Результати порівняння двох фрагментів тексту одного автора зазначені на рисунку 2.14. Статистичні методи тут дають великий розбіг;

- Неможливість працювати з декількома мовами. Система ідентифікації авторства тексту призначена винятково для роботи з українською. Водночас, це є перевагою, оскільки інші альтернативи не пропонували можливість використання української, або ж пропонували лише обмежене застосування інструментів саме для цієї мови.

ІНДЕКСИ АТРИБУЦІЇ ТЕКСТУ



Рисунок 2.14 – Порівняння двох текстів

Отже, було розглянуто декілька вже існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи. Було проаналізовано такі додатки: JGAAP (open-source), NeoNeuro (кіпрський розробник) та TextAttributor (українській розробник).

Усі вищенаведені рішення мають велику кількість недоліків. Це відноситься як до технічної, програмної, алгоритмічної реалізації, а також до морально-етичних колізій, які можуть виникнути під час використання саме цього програмного забезпечення.

Основні недоліки, виявлені під час дослідження, є наступними: використання винятково статистичних методів оцінки ідентифікації авторства тексту, потреба у підготовці великих корпусів текстів з боку користувача, застарілий графічний інтерфейс, який створює багато труднощів під час роботи з програмними комплексами (іноді не зрозуміло, що потрібно, де потрібно натискати, як відреагувала програма на дії користувача, які виникають проблеми, та як їх усунути для продовження роботи), пропрієтарне програмне забезпечення (звісно, це не відноситься до JGAAP, проте інші програми мають закритий вихідний код і подивитись його немає можливості; це створює такі ризики, що можуть призводити до погіршення стану кібергігієни, до загрози кіберінфраструктурі, а також постає питання етичності використання таких інструментів, які можуть бути на службі лише правоохоронних органів та корпорацій (бо тільки вони зможуть платити гроші за користування цим ПЗ), застарілі мови програмування та фреймворки, що використовувалися для створення додатків.

Надзвичайно серйозною проблемою окремо можна виділити питання етичності застосування цього програмного забезпечення. Багато з них не мають відкритий вихідний код, і дізнатись, як працює програма, як вона оцінює уривки тексту, можна лише за допомогою реверс-інженерінгу. Безумовно, наприклад, українська програма TextAttributor дозволяє користуватись і фізичним особам своїм функціоналом. Проте, якщо ця технологія стане комерційною, право у користувачів на використання може зникнути. Єдиним інструментом, який не має такої проблеми, є JGAAP.

Найкращою програмою серед вищенаведених можна назвати, певно, JGAAP. Вона має зрозумілий, хоча й простий інтерфейс, дозволяє вибирати методи та алгоритми, яким чином аналізувати текст. Дає змогу проводити обчислення локально за допомогою власних ресурсів CPU, тобто дані не йдуть на сервер. Вихідний код є відкритим. До сих пір програма підтримується ентузіастами і має власний репозиторій на Github. Це відрізняє JGAAP від двох

інших альтернатив, одна з яких взагалі має дивне офшорне походження, а інша не має власного репозиторію на платформі GitHub.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Перед створенням власної системи ідентифікації авторства тексту потрібно визначити засоби для побудови системи: мови програмування, які будуть використовувати, фреймворки, програмні інструменти, а також бібліотеки та допоміжне ПЗ. Окремо потрібно звернути увагу на те, що буде розділено програмне забезпечення та апаратне забезпечення. Оскільки створення додатку буде пов'язане із тренуванням нейронних мереж, то обчислювальні ресурси апаратного забезпечення дуже важливі. І це нижче буде більш детально обговорено.

До переліку, що потрібно визначити, належить:

- Середовище програмування, редактори, програмні розширення, плагіни та способи модифікації програмного забезпечення для роботи із вихідним кодом;
- Апаратне забезпечення, на базі якого буде проводитись навчання моделі нейронної мережі. До цього можна віднести як центральний процесор, так і графічний процесор, тобто CPU та GPU;
- Мова програмування;
- Фреймворк для машинного навчання та роботи із даними;
- Бібліотека для роботи із мережею, для роботи із локальними моделями на ПК;
- Інструментів для форматування вихідного коду (щоб зробити код одноманітним та стандартизованим);
- Хмарних інструментів для роботи із моделями через API-провайдерів.

Вим	Анк	№ докум.	Підпис	Дата

Незважаючи на те, що частина засобів є більш, а інша менш важливою, все одно потрібно визначитись із усім набором.

Безумовно, якщо брати до уваги середовище програмування, редактори вихідного коду та плагіни, що можна використовувати, слід відзначити наступні варіанти, а саме: Neovim та Visual Studio Code. Загалом не обов'язково брати саме VS Code, оскільки можна обмежитись форками або іншими редакторами, які базуються саме на ядрі програми від Microsoft.

Для кращої роботи із вищенаведеними програмами було встановлено обидва редактори плагіни та модифікації. Під час проведення розробки не було можливості обмежитись лише одним редактором. Це обумовлено тим, що ті плагіни та хоткеї, які використовуються в одному, не можна використовувати в іншому. Звісно, можна звернути увагу на те, що існують розширення, які дозволяють емулювати Vim у редакторі VS Code [14]. Але виникає проблема, коли декілька біндів конфліктують один з одним. Тому коли була можливість працювати із чистими файлами формату PY, варто застосовувати редактор від спільноти (Neovim), а коли потрібно було працювати із файлами Jupyter Notebook, наприклад, то використовувати додаток від Microsoft (VS Code), оскільки там значно краще реалізована підтримка цієї платформи.

Для покращення користувацького досвіду у Neovim можна використати збірку NvChad [15]. Завдяки тому, що у цьому редакторі коду для конфігурації плагінів використовується мова Lua, то проводити налаштування легко.

Для конфігурації VS Code було використано графічний інтерфейс користувача, який надає цей редактор коду. Обрано плагіни, які мають відкритий вихідний код, тобто open-source. Серед них можна виділити, по-перше, Pyright [16], по-друге, Jupyter Notebook Plugin [17].

Наступною важливою віхою є вибір апаратного забезпечення, яке можна буде використовувати для навчання нейронної мережі. Слід зауважити,

що у роботі було прийнято рішення відмовитись від використання статистичних методів аналізу, тобто програма не буде реалізовувати класичні алгоритмічні методи. Це обумовлено тим, що вони не дають коректного результату, якщо текст має складний стиль та екзотичні семантичні ознаки.

Для навчання нейронної мережі, безумовно, найкращим вибором є GPU [18]. Більшість задач найкраще усього виконується саме на такому типі процесору. Привабливим виглядає використання TPU, тобто тензорного процесору, але тут виникає проблема, тому що TPU, розрахованих на масового користувача, просто немає. До того ж, важливо згадати про те, що багато сучасних моделей TPU розробляються компанією Google, а вона не продає навіть юридичним особам такі пристрої. Тобто навіть якщо були б фінанси та юридична особа, яка є офіційно зареєстрованою, то не можна було б отримати це через політику доступності. Є можливість використовувати TPU через Google Cloud [19], проте у такому випадку впливає питання етичне. Наскільки правильно тренувати власну систему ідентифікації авторства тексту, тобто модель для неї, використовуючи потужності Google Cloud? У такому випадку компанія може більше дізнатися про продукт і оскільки вона співпрацює із спеціальними службами США, це може становити загрозу для конфіденційності інформації. Також це може призвести до надмірної залежності від хмарних обчислень. Тому якщо не буде фінансів або ж юридичної доступності цих інструментів (наприклад, через санкції, які погрожував увести діючий на момент написання бакалаврської роботи Президент США Д. Трамп «проти усіх сторін конфлікту» [20]), можуть відбутись проблеми із розробкою та використанням. Враховуючи характер системи та можливі ресурси, які потрібно буде витратити на навчання, відеокарти із бюджетного сегменту для персональних комп'ютерів буде достатньо. Головна умова – це те, щоб відеокарта підтримувала сучасну версію технології CUDA. На даний момент це найкраща технологія обчислень за допомогою GPU для задач, не пов'язаних із комп'ютерною графікою.

Водночас, подібна позиція точно відображає домінацію компанії NVIDIA у цій сфері. Порівняння функціоналу технологій знаходиться у таблиці 2.1.

Окремо слід зауважити, що платформи, завдяки яким проводяться обчислення, тісно пов'язані із апаратним забезпеченням. Можна сказати, що технології обчислень залежать від апаратного забезпечення. Принаймні, такий підхід вірний щодо CUDA (через безпосередню залежність від GPU NVIDIA).

Інші платформи підтримують обчислення на графічних прискорювачах від інших виробників, однак наразі застосовуються переважно експериментальні підходи, що розробляються, мають велику кількість багів, а також потребують доопрацювань (скажімо, для запуску обчислень за допомогою ROCm на відеокартах від NVIDIA). Жорстка прив'язка до конкретної апаратної бази виникла внаслідок специфіки еволюції мікропроцесорної техніки. Архітектура графічних процесорів є вкрай спеціалізованою, тому кожен виробник оптимізує апаратні блоки згідно з власним концептуальним баченням паралельних обчислень. У випадку з компанією NVIDIA спостерігається багаторічна стратегія створення закритої, проте надзвичайно інтегрованої екосистеми. Архітектурні рішення розвивалися паралельно з програмним стеком, що дозволило досягти максимальної синергії між обладнанням та кодом. Конкурентним підприємствам доводиться не просто створювати власні аналоги, але й намагатися адаптувати розробки під стандарти, які де-факто вже були встановлені лідером ринку.

Водночас у науково-дослідному середовищі можна побачити постійні спроби створення універсальних рішень. Наприклад, стандарт OpenCL від початку розроблявся як кросплатформенний інструмент, призначений для роботи на будь-якому обладнанні – від центральних процесорів до спеціалізованих прискорювачів. Однак на практиці виявилося, що написання дійсно ефективного програмного коду вимагає глибокого розуміння особливостей конкретного напівпровідникового кристала.

Вим	Анк	№ докум.	Підпис	Дата

Таблиця 2.1 – Порівняння технологій обчислень за допомогою GPU

КРИТЕРІЙ	CUDA (NVIDIA) [21]	ROCM (AMD) [22]	ONEAPI (INTEL) [23]
Ліцензійний режим	Пропріетарний SDK та драйвери	Open-source ядро з компонентами пропріетарних драйверів	Open-source компоненти з пропріетарними модулями
Підтримка PyTorch	Офіційна, доступні бінарні збірки та контейнерні образи	Офіційна; upstream; релізи ROCm і PyTorch можуть мати незалежні цикли.	Підтримка через oneDNN/oneAPI бекенди;
Сумісність із Jupyter Notebook	Повна інтеграція в локальні середовища та контейнерні образи; широко застосовується у наукових робочих процесах.	Підтримується; рекомендовано використовувати офіційні Docker образи ROCm для мінімізації проблем з драйверами та правами доступу.	Працює для CPU-прискорення та експериментально для Intel GPU; інтеграція в Jupyter
Сумісність із операційними системами	Linux, Windows; macOS – обмежено через апаратні/платформні обмеження.	Переважно Linux; офіційна підтримка Windows обмежена або експериментальна.	Linux, Windows; функціонал для різних апаратних бекендів залежить від драйверів та релізів oneAPI.
Продуктивність у задачах AI/ML	Висока; оптимізована екосистема (cuDNN, NCCL, TensorRT) для тренування та інференсу.	Конкурентна на сертифікованих GPU; продуктивність залежить від maturity MIOpen та сумісності драйверів.	Сильна на CPU завдяки oneDNN; GPU-прискорення залежить від конкретної реалізації та драйверів.
Екосистема та інструментарій	Розвинена: бібліотеки для лінійної алгебри, оптимізації та контейнерні образи.	MIOpen, ROCm Docker, HIP як шар сумісності з CUDA; екосистема зростає.	oneAPI, DPC++, oneDNN; фокус на уніфікації обчислень між CPU/GPU/FPGA.

Вибір графічного процесору залежить від того, чи він підтримує технологію CUDA чи ні, а також його доступність.

Наступним важливим етапом стане вибір програмного забезпечення для реалізації системи ідентифікації авторства тексту.

Наразі існує декілька варіантів для написання вихідного коду програми, а саме: C++, Python або Java. Останній варіант не є прийнятним, оскільки Java була популярною у минулому. Водночас, зараз вона втратила актуальність і залишається лише вибір із перших двох мов.

Вибір C++ є привабливим, оскільки ця мова має високу продуктивність, дає контроль над пам'яттю, має велику кількість бібліотек та фреймворків, де вже є необхідний функціонал. А також саме діалект цієї мови використовується на платформі CUDA (тому якщо буде потреба щось модифікувати, то можна буде не змінювати мову програмування). Але, водночас, C++ є надзвичайно складною мовою, особливо якщо ми беремо обчислення на базі графічного процесору, а також взаємодію із CUDA. Програма має ризики витоку даних, проблем з пам'яттю, дуже складно працювати із вказівниками та робити об'єктно-орієнтований підхід. Загалом, кваліфікація програміста повинна бути неймовірно високою для використання саме цього інструменту. Для написання системи, яка була визначена у цій роботі, це надлишковий підхід до вирішення задачі. Модифікувати безпосередньо файли бібліотек немає необхідності, бо увесь функціонал уже там реалізований. Тому потрібні більш практичні та прості інструменти для роботи із нейромережами. Саме це дає використання мови Python. І про це буде йти мова далі.

Мова Python має зручний синтаксис, простий для написання вихідного коду та подальшої підтримки, а також модифікації. Має велику кількість бібліотек та фреймворків, активне ком'юніті, чудову документацію. Це стосується як самої мови, так і супутніх інструментів. Багато програмістів використовують інструменти для форматування коду, і для них теж є

документація. А також це найбільш популярна мова серед розробників із машинного навчання. Популярні фреймворки на Python вже використовують платформу CUDA, реалізують ефективні обчислення, роботу з пам'яттю і з графічним процесором. Більше інформації зазначено у таблиці 2.2. Там порівнюються різні фреймворки.

Таблиця 2.2 – Порівняння фреймворків для машинного навчання

Критерій	Tensorflow/ Keras [24] [25]	PyTorch [26]	JAX [27]	MXNet [28]
Організація-розробник	Google (США)	Meta AI / Fair (США)	Google DeepMind (США)	Apache Foundation / Amazon (США)
Ліцензія	Apache 2.0	BSD-3-Clause	Apache 2.0	Apache 2.0
Актуальна версія	TF 2.17 / Keras 3.x	PyTorch 2.4	JAX 0.4.x	MXNet 1.9 (припинено розробку)
Розмір	pip install tensorflow (~500 МБ)	pip install torch (~750 МБ)	pip install jax (~100 МБ)	pip install mxnet (~300 МБ)
Підтримка операційних систем	Windows, Linux, macOS	Windows, Linux, macOS	Linux (повна), macOS / Win – частково	Linux, Windows (macOS – обмежено)
Режим CPU-only	Так	Так	Так	Так
Підтримка GPU NVIDIA (CUDA) на локальній машині	Так (CUDA 11–1*)	Так (CUDA 11–1*, пріоритетна)	Так (CUDA через XLA)	Так (CUDA 10–11)
Швидкість виконання інференсу на CPU	Висока (TF Lite / XLA)	Висока (TorchScript)	Дуже висока (JIT/XLA-компіляція)	Середня

У результаті було обрано фреймворк PyTorch. Незважаючи на те, що він має більший розмір, аніж інші варіанти, проте він адаптований під більшість операційних систем, підтримує сучасну версію CUDA, має

оптимізовані обчислення як на CPU, так і на GPU, і має зручну документацію.

Для роботи із неочищеними даними, із мережею потрібно підібрати програмні інструменти. Це повинні бути бібліотеки для мови програмування Python. Вони повинні підтримувати версію інтерпретатора вище 3.12 включно. Серед них було обрано наступні: `ollama`, `requests`, `pathlib`. Надалі буде більш детальний опис кожного із цих інструментів.

`Ollama` – це інструмент для локального запуску великих мовних моделей без необхідності підключення до зовнішніх API [29]. Він дозволяє завантажувати та виконувати моделі безпосередньо на машині розробника, що є критично важливим для задач, де передача даних у хмару є небажаною або неможливою. Підтримує Python 3.12 і вище, має офіційний клієнт у вигляді бібліотеки `ollama` [30], яка встановлюється через `pip`.

Взаємодія із моделями відбувається через простий інтерфейс: достатньо передати текстовий запит і отримати відповідь у вигляді рядка або потокового виводу. Це зручно, коли потрібно обробити велику кількість текстових фрагментів – наприклад, для аналізу авторського стилю. Бібліотека підтримує асинхронний режим роботи, що дозволяє не блокувати основний потік виконання під час генерації.

Варто зазначити, що `Ollama` потребує окремого серверного процесу, який запускається на локальній машині. Це незначне ускладнення інфраструктури, однак воно компенсується повним контролем над середовищем виконання та відсутністю залежності від стороннього провайдера. Для цілей даної роботи це є перевагою, а не недоліком. Водночас, нещодавно був запущений сервіс `Ollama Cloud` [31], який дозволяє використовувати хмарні обчислення, якщо немає локальних ресурсів.

Бібліотека `requests` – стандарт де-факто для виконання HTTP-запитів у Python. Вона присутня у переважній більшості проєктів, де є потреба у взаємодії із зовнішніми або локальними веб-сервісами. Підтримує всі актуальні версії інтерпретатора, включно із 3.12. Встановлюється через `pip`

однією командою.

API бібліотеки влаштований так, що типовий запит записується в один рядок коду. GET, POST, передача заголовків, обробка відповіді – усе це реалізовано інтуїтивно та без зайвого синтаксичного шуму. Для отримання текстових даних із зовнішніх джерел `requests` є гарним вибором.

Окремо варто відзначити надійність бібліотеки: вона активно підтримується, має вичерпну документацію і фактично не змінює свій інтерфейс між версіями. Це означає, що код, написаний сьогодні, буде працювати без модифікацій через рік. Для дипломної роботи, де відтворюваність результатів є обов’язковою умовою, це суттєво.

`Pathlib` є стандартною бібліотекою Python для роботи з файловою системою [32]. Вона входить до складу інтерпретатора починаючи з версії 3.4, тому встановлення не потрібне. Надає об’єктно-орієнтований інтерфейс до шляхів файлів і директорій, що є зручнішим за роботу із рядками у старому стилі через `os.path`.

Основна перевага `pathlib` – це читабельність коду. Операції на кшталт пошуку файлів за маскою, перевірки існування, зчитування вмісту або побудови шляху записуються компактно і без нагромодження вкладених викликів функцій. Це особливо помітно під час обробки корпусів текстів, де потрібно масово читати файли із певної директорії.

Бібліотека є кросплатформеною: один і той самий код коректно виконується на Windows, Linux і macOS без жодних модифікацій. Враховуючи, що розробка та подальше розгортання системи можуть здійснюватися у різних операційних середовищах, це виключає цілий клас потенційних помилок, пов’язаних із різними роздільниками шляхів та особливостями файлових систем. Водночас, дуже важливим є той факт, що існує гарна документація до цього програмного модулю: вона дозволяє визначити основні класи, методи, поля, що потрібні для написання коду.

2.3 Розгорнута постановка завдання

Загалом, завдання можна визначити наступним чином: спроектувати систему ідентифікації авторства тексту, яка буде базуватись на програмних інструментах, бібліотеках, фреймворках та платформах Python, PyTorch, CUDA. Вона буде використовувати нейронну мережу для виконання поставленого завдання. Не будуть застосовуватись статистичні методи ідентифікації, а також буде розгорнуто інфраструктуру, яка дозволить не тільки натренувати нейронну мережу, але й використовувати її у подальшому. Можна застосовувати також декотрі класичні методи звичайного машинного навчання, але потрібно зосередити свої зусилля винятково на більш прагматичних підходах.

Програмне забезпечення повинно запускатись на операційних системах сімейства Linux. Найбільш пріоритетним є можливість використання та коректної роботи на Ubuntu 24.04 LTS.

Вихідний код повинен бути відформатований за допомогою інструмента Ruff.

В залежності від конкретних результатів та патернів, що будуть використовуватись, під них будуть адаптуватись параметри та налаштування.

Вим	Анк	№ доквм.	Підпис	Дата

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Система ідентифікації авторства тексту базується на принципі порівняльного аналізу стилістичних ознак, а не за принципом класифікації. де кожному автору присвоюється окрема мітка. Це головне, ключе архітектурне рішення, оскільки класифікаційний підхід вимагає, щоб усі потенційні автори були заздалегідь відомі системі на етапі навчання, що робить його непридатним для реальних сценаріїв використання, коли потрібно перевірити тексти, автори яких були відсутні під час навчального етапу.

Натомість обрана архітектура за принципом сіамської мережі навчається розпізнавати, виділяти та порівнювати стилістичні шаблони у довільних парах текстових фрагментів. Тобто система здатна відповісти, якщо дати два уривки тексту, чи написані вони одним автором, чи ні. І для цього не потрібно щоб автор був відомий на момент навчання.

Подібна постановка задачі – це верифікація авторства тексту і є значно ближчою до реальних практичних викликів, аніж закритий класифікаційний підхід.

Водночас, оскільки стиль автору проявляється на декількох рівнях, а саме: лексичному виборі, частоті вживання слів, синтаксичних конструкцій, ритмічних закономірностей, система використовує двонаправлену рекурентну нейронну мережу із довгостроковою пам'яттю LSTM [33]. Вона здатна враховувати контекст в обох напрямках, тобто як справа, так і зліва від кожного слова. Результатом аналізу є числовий вектор фіксованої розмірності, який являється представленням стилістичного профілю тексту. Два вектори, отримані для кожного із текстових фрагментів, аналізуються класифікатором, який оцінює рівень їх подібності та формує оцінку авторства. Для роботи із

фрагментами, що перевищують допустиму довжину, відбувається процес їх розбиття на менші уривки.

Загалом, уся система – це конвеєр із механізмами кешування, комунікації та верифікації завдяки чому повторні запуски не потребують повторного обчислення вже виконаних операцій, а окремі компоненти системи можуть бути замінені або модифіковані незалежно один від одного (або ж із мінімальними зусиллями).

Розробка математичної моделі базується на використанні сіамської мережі. Використання саме такої архітектури обумовлено тим, що завдяки такому підходу нейронна мережа навчається вивчати стилістичні ознаки різних текстів або ж фрагментів та порівнювати їх між собою, а не просто орієнтуватись на конкретних авторів. Водночас, саме таке рішення є оптимальним із урахуванням обчислювальних ресурсів апаратного забезпечення. Варто дослідити, розібрати та проаналізувати математичну модель, що була спланована.

Картинка математичної моделі зображена на рисунку 3.1. Вона складається із дев'яťох блоків. Дії відбуваються послідовно, переходячи із одного блока в інший (згідно із індексом, який є натуральним числом). Умовно можна розділити мат. модель на три рівня. Спочатку йде препроцесинг, ініціалізація; наступний рівень характеризує стилістичний аналіз тексту; останній рівень характеризує вихідну класифікацію (рішення чи є фрагменти текстами одного автора), метод зворотного поширення помилки, метрики навчання, оптимізаційні критерії та булеве значення, чи є автор один і той же (позначений спеціальною літерою P). Архітектура нейронної мережі є двонаправленою, тому це додає більше контексту для винесення вердикту, але при цьому збільшує потребу в ресурсах. Значення 0,65 для оцінки того, чи автор один, є найбільш оптимальним із погляду на емпіричні дані, які отримані для валідаційних й тренувальних даних. В залежності від наборів для навчання це значення може змінюватись.

Математична модель системи ідентифікації

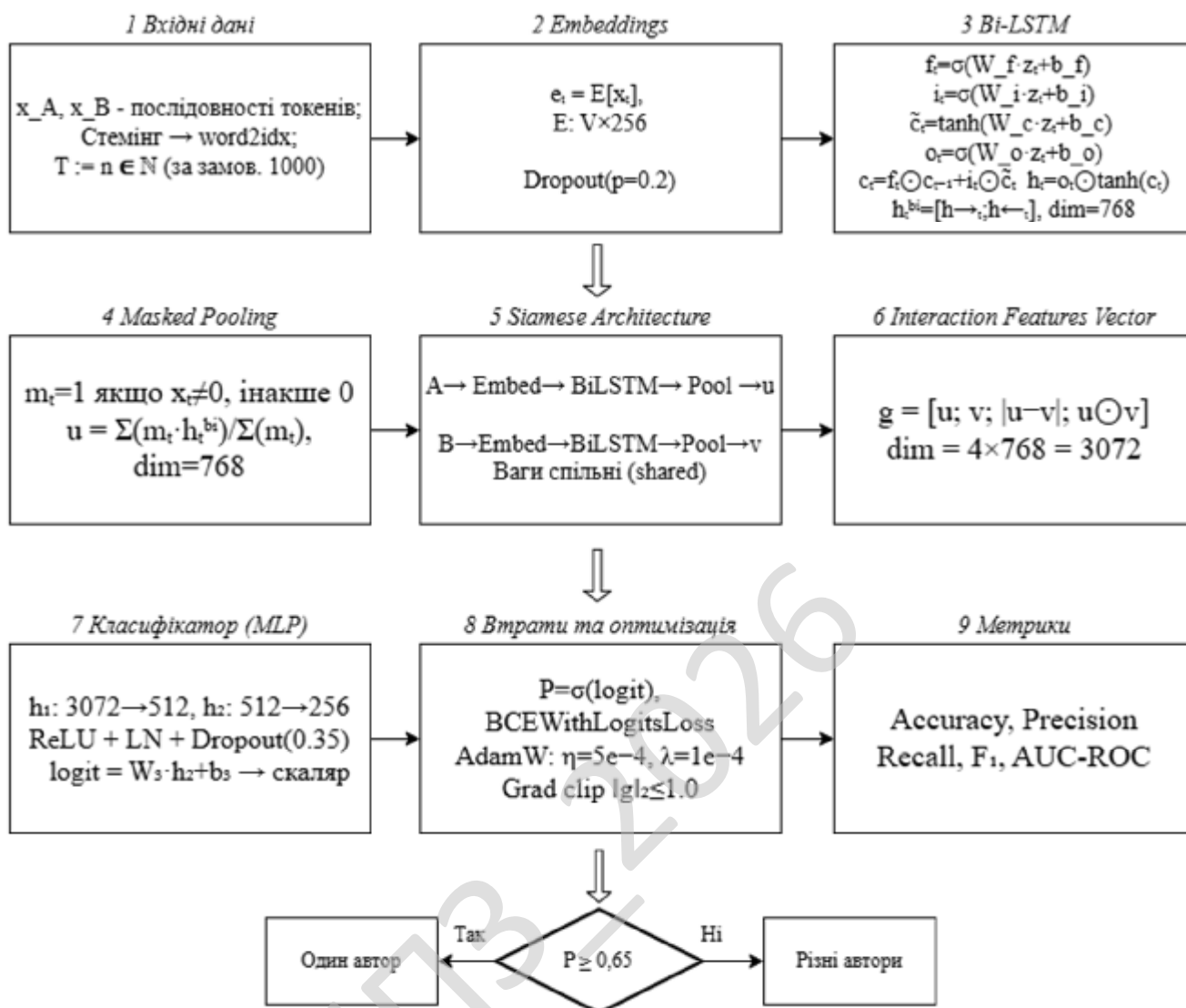


Рисунок 3.1 – Математична модель системи ідентифікації

На вхід системи подаються два тексти. Для подальшого аналізу називатимемо їх Текст А та Текст В. Кожен із них повинен пройти процедуру токенізації. Після токенізації застосовується стемінг, реалізований шляхом суфіксного відсікання: система перевіряє, чи закінчується слово на один із визначених українських суфіксів і, якщо цей суфікс є у списках, відсікає його, залишаючи корінь довжиною не менше ніж два символи. Суфікси відсортовані за довжиною, що гарантує відсікання спочатку більших суфіксів (якщо такі знайдені), а вже потім коротких. Опісля кожен токен співставляється числовому індексу (за допомогою спеціального словника). У словнику

заздалегідь зарезервовані два спеціальних символи: $\langle PAD \rangle$ 0 та $\langle UNK \rangle$ 1. Результатом є кортеж цілих чисел, зазначений у формулі 3.1.

$$x = (x_1, x_2, \dots, x_T), \quad (3.1)$$

де T – це фіксована довжина послідовності (1000 символів).

Надалі потрібно перетворити числові індекси в просторові тензори, що називаються ембедінгами. Вони будуть відображати лінгвістичні ознаки слів.

Якщо говорити більш конкретно: Для цього використовується матриця E , яка зазначена у формулі 3.2.

$$E \in R^{V \times d_e}, \quad (3.2)$$

де E – це матриця ембедінгів;

R – множина дійсних чисел;

V – це розмір словника;

d_e – розмір embeddings.

Слід зауважити, що $\langle PAD \rangle$ є нульовим вектором та не змінює значення своїх компонентів під час навчання. Він використовується тільки для заповнення фрагменту тексту, якщо його довжина менше 1000 (узагалі, система спроектована таким чином, що користувач може змінювати значення довжини, але було обране саме таке значення). Під час тренування за допомогою dropout із імовірністю 0,2 може відбутись обнуління частини елементів вектора.

Вищезазначена матриця є навчальним параметром моделі: на початку він ініціалізується випадковими значеннями, а під час навчання градієнти та коригують вектори слів таким чином, щоб мінімізувати функцію втрат.

Архітектура нейронної мережі базується на LSTM. Цей вибір обумовлений тим, що RNN має проблему вибухання градієнтів або ж їх затухання, тому що на кожній ітерації навчання відбувається множення матриці із вагами. Для довгих послідовностей (у цьому випадку довжиною у

тисячу) звичайна рекурентна нейронна мережа не в змозі аналізувати довгострокові залежності (що представляє собою текст). LSTM вирішує цю проблему завдяки введенню окремого стану c_t , який оновлюється через адитивну операцію, а не через рекурентне множення матриць.

Для кожного кроку від 1 до T обчислюються наступні величини:

Input Vector (вхідний вектор) – це векторизоване представлення поточного токена із урахуванням контексту. Він зазначений на рисунку 3.3.

$$z_t = [h_{t-1}; e_t], \quad (3.3)$$

де z_t – об'єднаний вхідний вектор;

h_{t-1} – прихований стан на попередньому кроці;

e_t – вектор вкладення поточного токена;

$[\ ; \]$ – операція склеювання двох векторів в один; результат буде мати розмірність $d_h + d_e = 384 + 256 = 640$.

На основі цього вектору обчислюються нижчезазначені вентиля (англ. gates).

Forget Gate (вентиль забування) – цей вентиль визначає, яку частину попереднього стану комірки зберегти, а яку "забути". Він зазначений у формулі 3.4.

$$f_t = \sigma(W_f \cdot z_t + b_f), \quad (3.4)$$

де f_t – вектор значень на кроці t , кожен елемент знаходиться у проміжку $(0, 1)$, розмірність вектору $d_h = 384$;

σ – сигмоїдна функція активації, що перетворює довільне дійсне число на значення у проміжку $(0, 1)$;

W_f – навчальна матриця ваг вентиля розмірності $d_h \times (d_h + d_e) = 384 \times 640$;

z_t – об'єднаний вхідний вектор;

b_f – вектор зміщення розмірності $d_h = 384$.

Input Gate (вхідний вентиль) – цей вентиль вирішує, які нові дані записати у комірку пам'яті. Він зазначений у формулі 3.5.

$$i_t = \sigma(W_i \cdot z_t + b_i), \quad (3.5)$$

де i_t – вектор значень вентилля на кроці t , кожен елемент знаходиться у проміжку $(0, 1)$, розмірність вектору $d_h = 384$;

W_i – матриця коефіцієнтів вхідного вентилля розмірністю 384×640 ;

b_i – вектор зміщення розмірності $d_h = 384$.

Cell Candidate (кандидат у комірку пам'яті) – цей вентиль формує новий вміст пам'яті, що може бути записаний у комірку. Він зазначений у формулі 3.6.

$$\tilde{c}_t = \tanh(W_c \cdot z_t + b_c), \quad (3.6)$$

де $\tilde{c}_t$ – це вектор для оновлення стану комірки на кроці t , кожен елемент якого знаходиться у проміжку $(-1, 1)$, розмірність $d_h = 384$;

$\tanh(x)$ – гіперболічний тангенс, функція активації;

W_c – матриця коефіцієнтів cell candidate розмірності $d_h \times (d_h + d_e) = 384 \times 640$;

b_c – вектор зміщення розмірності $d_h = 384$.

Output Gate (вихідний вентиль) – це вентиль, який контролює, яка частина стану комірки потрапить на вихід. Він зазначений на рисунку 3.7.

$$o_t = \sigma(W_o \cdot z_t + b_o), \quad (3.7)$$

де o_t – це вектор значень вихідного вентилля на кроці t , кожен елемент знаходиться у проміжку $(0, 1)$, розмірність $d_h = 384$;

W_o – це матриця коефіцієнтів вихідного вентилля розмірності 384×640 ;

b_o – вектор зміщення вихідного вентилля.

Cell State Update (оновлення комірки пам'яті) – це операція оновлення

стану комірки пам'яті, що зберігає інформацію про усі попередні токени від початку послідовності до кроку t . Вона зазначена у формулі 3.8.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad (3.8)$$

де f_t – це вектор вентиля забування (визначає, яка частина попереднього стану комірки пам'яті зберігається: значення від 0 до 1, де чим ближче до 1, тим краще запам'ятовується);

$\odot$ – операція поелементного добутку Адамара, де кожен елемент першого вектора множиться на відповідний елемент другого;

c_{t-1} – стан комірки пам'яті на попередньому кроці;

$+$ – поелементне додавання двох векторів однакової розмірності;

i_t – вектор вхідного вентиля;

c_t – вектор-кандидат на новий вміст пам'яті.

Hidden State (прихований стан) – це прихований на кроці t розмірності $d_h = 384$. Це основне вихідне значення LSTM-комірки у даній фазі. Він зазначений у формулі 3.9.

$$h_t = o_t \odot \tanh(c_t), \quad (3.9)$$

Надалі слід розглянути двонаправленість. Це є важливою частиною архітектури нейронної мережі. Стандартна LSTM обробляє фрагмент лише в одному напрямку (якщо тривіалізувати, то зліва направо). Для аналізу стилістики тексту це неприйнятне як для дослідження обмеження, тому що контекст слова може залежати не тільки від того, що було до нього у тексті, але й від того що буде після. Тому у системі буде використовуватися Bidirectional LSTM, уперше запропонований у роботі Шмідхубера [34]. Опис цього підходу можна побачити у формулі 3.10.

$$h_t^{bi} = [h_t^{\rightarrow}; h_t^{\leftarrow}], \quad (3.10)$$

де h_t^{bi} – це двонаправлений прихований стан на кроці t розмірності $2 \times$

$$d_h = 2 \times 384 = 768;$$

$h_t^{\rightarrow}$ – це прихований стан прямого LSTM;

$h_t^{\leftarrow}$ – це прихований стан зворотнього LSTM.

Таким чином, результатом роботи BiLSTM є матриця H розмірності $T \times 768$. Кожен рядок цієї матриці містить представлення відповідного токена із урахуванням контексту. Це зазначено у формулі 3.11.

$$H = \begin{pmatrix} h_1^{bi} \\ h_2^{bi} \\ \vdots h_T^{bi} \end{pmatrix} \in R^{T \times 768}, \quad (3.11)$$

Опісля проведення вищезазначених маніпуляцій потрібно якимось чином агрегувати усі T векторів. Для цього потрібно взяти середнє значення. Але постає проблема символів-заповнювачів $\langle PAD \rangle$, тому що вони створюють «шуми». Для такого випадку винайшли Masked Average Pooling.

Перш ніж розглядати це поняття слід визначити, що таке маска. Маска визначається у формулі 3.12.

$$m_t = \mathbb{1}(x_t \neq 0), \quad (3.12)$$

де m_t – значення маски для позиції t (повинно дорівнювати 0 для символів заповнення та 1 для реальних символів);

$\mathbb{1}$ – індикаторна функція, яка повертає 1, якщо умова всередині дужок є істинною та 0 в усіх інших випадках;

x_t – це числовий індекс на позиції t (індекс 0 для символів $\langle PAD \rangle$);

$x_t \neq 0$ – умова істинності.

Середнє агреговане значення, тобто Masked Mean, зазначено у формулі 3.13 нижче.

$$u = \frac{\sum_{t=1}^T m_t \cdot h_t^{bi}}{\sum_{t=1}^T m_t}, \quad (3.13)$$

де u – вектор представлення усього тексту розмірності 768;

T – максимальна довжина фрагменту тексту;

m_t – значення маски для позиції t ;

h_t^{bi} – двонаправлений прихований стан на кроці t .

Обидва тексти (текст А та текст В) обробляються одним кодувальником [35]. Спільні ваги забезпечують інваріантність відносно вхідних текстів: помінявши місцями текст А та текст В, все одно отримують ті ж самі вектори u та v . Слід зазначити, що на рівні кодувальника симетрія є повною, проте вектор взаємодії (описаний нижче у розділі) містить компоненту прямої конкатенації $[u; v]$, яка не є інваріантною до перестановки ($[u; v] \neq [v; u]$ у загальному випадку). Натомість компоненти абсолютної різниці $|u - v|$ та поелементного добутку $u \odot v$ є симетричними за побудовою. Отже, класифікатор може навчитись як симетричним, так і несиметричним (асиметричним) ознакам. При цьому на практиці основний внесок роблять саме симетричні характеристики, оскільки якщо тексти схожі, то вони повинні мати подібні стилістичні ознаки, тобто бути у декотрому розумінні симетричними. Саме це можна назвати проявом схожості обох фрагментів.

Після отримання векторів представлень обох текстів їх потрібно якимось чином порівняти. Найбільш простим варіантом дій у такому випадку це взяти якусь одну характеристику, наприклад, косинусну подібність. Але у такому випадку виникає проблема втрати цінної інформації про вхідні данні. Замість цього слід зробити Interaction Features Vector (вектор характеристик взаємодії або ж просто вектор взаємодії) [36] [37]. Він зображений у формулі 3.14.

$$g = [u; v; |u - v|; u \odot v] \in R^{3072}, \quad (3.14)$$

де g – вектор взаємодії розмірності $768 + 768 + 768 + 768 = 3072$;

u – вектор представлення тексту А;

v – вектор представлення тексту B ;

$|u - v|$ – абсолютна різниця: для кожної координати i обчислюється $|u_i - v_i|$, розмірності 768;

R^{3072} – вектор дійсних чисел розмірності 3072.

Опісля вектор взаємодії g подається на багат шаровий перцептрон, що складається із двох hidden layers та одного output layer. Він зазначений у формулі 3.15.

$$h_1 = W_1 \cdot g + b_1, \quad (3.15)$$

де h_1 – вихід першого прихованого шару до нормалізації та активації, розмірності 512;

W_1 – матриця коефіцієнтів першого шару розмірності 512×3072 ;

g – вхідний вектор взаємодії;

b_1 – вектор зміщення розмірності 512.

До результату h_1 послідовно застосовуються: нормалізація шару, функція активації ReLU [38], Dropout. Вони зазначені у формулах 3.16, 3.17, 3.18 відповідно. Слід також зауважити, що потім значення із цього шару будуть передані у другий. І там буде теж аналогічна структура. Але про це мова буде йти далі. Наразі розберемо математичні нюанси, пов'язані саме із цією частиною математичної моделі.

$$\text{LN}(x) = \gamma \cdot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \quad (3.16)$$

де x – вхідний вектор (у даному випадку h_1);

μ – середнє значення, що обчислюється за $\mu = \frac{1}{d} \sum_{i=1}^d x_i$;

σ^2 – дисперсія по координатах одного прикладу, що обчислюється за $\sigma^2 = \frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2$;

d – розмірність вхідного вектору;

ϵ – мала константа, що додається для уникнення ділення на нуль;

β – вектор зміщення розмірності d , що ініціалізується нулями.

$$\text{ReLU}(x) = \max(0, x) \quad (3.17)$$

де $\max(0, x)$ – операція вибору максимуму з двох значень: нуля та x .

$$h'_1 = \text{Dropout}(\text{ReLU}(\text{LN}(h_1))) \quad (3.18)$$

де $\text{Dropout}(x)$ – функція, що із заданою ймовірністю p обнуляє кожен елемент, при цьому масштабуючи решту елементів множником $\frac{1}{1-p}$.

Другий шар має аналогічну структуру, що зазначена у формулі 3.19 та 3.20.

$$h_2 = W_2 \cdot h'_1 + b_2 \quad (3.19)$$

де h_2 – вихід лінійного перетворення другого шару до нормалізації та активації розмірності 256;

W_2 – матриця коефіцієнтів другого шару розмірності 256×512 ;

h'_1 – вихід першого шару після нормалізації, ReLU та Dropout розмірності 512;

b_2 – вектор зміщення другого шару розмірності 256.

$$h'_2 = \text{Dropout}(\text{ReLU}(\text{LN}(h_2))) \quad (3.20)$$

де h'_2 – вихід другого прихованого шару після нормалізації, ReLU та Dropout, розмірності 256.

Вихідний шар наведений у формулі 3.21. Він видає скалярне значення, тобто дійсне число, що далі використовується для обчислення ймовірності.

$$\text{logit} = W_3 \cdot h'_2 + b_3, \quad (3.21)$$

де logit – скалярне значення, що далі використовується для обчислення ймовірності;

W_3 – матриця коефіцієнтів вихідного шару розмірності 1×256 ;

h'_2 – вихід другого прихованого шару після перетворень розмірності 256;

b_3 – скалярне зміщення вихідного шару.

Розгляну функцію втрат та оптимізацію. Основний спосіб перетворення логіту на ймовірність зазначений у формулі 3.22. Він використовує лише сигмоїдну функцію, яка вже була зазначена раніше, але все одно через міркування точності викладної математичної моделі це вказується. Опісля спробуємо розглянути функцію втрат BCEWithLogitsLoss.

$$P = \sigma(\text{logit}) = \frac{1}{1+e^{-\text{logit}}}, \quad (3.22)$$

де P – ймовірність того, що два тексти належать одному автору, значення у проміжку $(0,1)$;

$\sigma(x)$ – сигмоїдна функція;

logit – скалярний вихід класифікатора (будь-яке дійсне значення);

e – число Ейлера.

Для обчислення функції втрат використовується формула, зазначена у 3.23. Обчислюється безпосередньо із логіту.

$$\mathcal{L} = \max(\hat{l}, 0) - \hat{l} + \ln(1 + e^{-\hat{l}}), \quad (3.23)$$

де $\mathcal{L}$ – значення функції втрат для одного прикладу, невід'ємне число;

$\hat{l}$ – значення логіту (він отриманий; змінене позначення);

y – мітка класу, яка приймає значення $y = 1$, якщо тексти належать одному автору, і $y = 0$ в іншому випадку;

$\max(x)$ – функція, що повертає $\hat{l}$ при $\hat{l} > 0$ та 0 інакше.

Використання вищенаведеної формули є виправданим, оскільки при дуже великих або дуже малих значеннях логіту наївні методи призводять до numerical overflow.

Для навчання на міні-батчах обчислюється середнє значення втрат по

усіх прикладах у батчі розміром B . Це зазначено у формулі 3.24.

$$\mathcal{L}_{batch} = \frac{1}{B} \sum_{j=1}^B \mathcal{L}_j, \quad (3.24)$$

де $\mathcal{L}_j$ – значення функції втрат для j -го прикладу у батчі;

B – розмір батчу.

Для оптимізації параметрів мережі використовується AdamW (тобто Adam із декомпонованою ваговою регуляризцією). Він зберігає exponential moving average of gradients – перший момент та exponential moving average of squared gradients – момент другого порядку. Подальші математичні нюанси виходять поза компетенції спеціальності, тому в даному випадку потрібно зафіксувати, що це просто покращує навчання. Формула AdamW зазначена у 3.25.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \quad (3.25)$$

де m_t – exponential moving average of gradients (момент першого порядку) на кроці оптимізації t ;

v_t – exponential moving average of squared gradients (момент другого порядку) на кроці оптимізації t ;

g_t – градієнт функції втрат по параметрах на поточному кроці;

$\beta_1 = 0,9$ – коефіцієнт експоненціального згладжування для першого моменту;

$\beta_2 = 0,999$ – коефіцієнт експоненціального згладжування для другого моменту.

Оскільки на початку навчання обидва моменти ініціалізуються нулями, застосовується bias correction (корекція зміщення). Вона зазначена у формулі 3.26.

$$\hat{m}_t = \frac{m_t}{1 - \frac{1}{t}}, \quad \hat{v}_t = \frac{v_t}{1 - \frac{1}{t}}, \quad (3.26)$$

де $\hat{m}_t$ – скоригований момент першого порядку;

Вим	Анк	№ докум.	Підпис	Дата

$\hat{v}_t$ – скоригований момент другого порядку;

$\beta_1^t - \beta_1$ у степені t (номер поточного кроку оптимізації);

$\beta_2^t - \beta_2$ у степені t ;

$(1 - \beta_1^t)$ та $(1 - \beta_2^t)$ – корекційні множники, що компенсують початкове зміщення до нуля; на перших кроках ці множники суттєво змінюють значення моментів, а з часом їхній вплив зменшується.

Оновлення коефіцієнтів (ваг) виконується за формулою зазначеною на 3.27.

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} + \lambda \theta_t \right), \quad (3.27)$$

де θ_t – вектор усіх навчальних параметрів мережі на кроці t ;

η – константа, що представляє собою темп навчання (розмір кроку оновлення);

$\hat{m}_t$ – скоригований момент першого порядку (напрямок оновлення);

$\hat{v}_t$ – скоригований момент другого порядку (масштаб оновлення по кожній координаті);

$\epsilon = 10^{-}$ – константа для запобігання діленню на нуль;

$\lambda = 10^{-}$ – коефіцієнт вагової регуляризації.

Головна відмінність модифікованої версії оптимізатора AdamW [39] від звичайного Adam із L2-регуляризацією полягає в тому, що вагова регуляризація $\lambda \theta_t$ тут decoupled, тобто є відокремленою від адаптивного оновлення і застосовується безпосередньо до параметрів. У звичайному алгоритмі Adam weight decay додається до градієнту перед обчисленням моментів, через що він масштабується адаптивним темпом навчання і діє нерівномірно для різних параметрів

Обчислюється глобальна L2-норма усіх градієнтів моделі [40], що зазначено у формулах 3.28 та 3.29.

$$|g|_2 = \sqrt{\sum_i g_i^2}, \quad (3.28)$$

де $|g|_2$ – евклідова норма вектора градієнтів, тобто довжина у евклідовому просторі;

g_i – це i -та компонента об'єднаного вектора усіх градієнтів моделі.

Якщо ця норма перевищує порогове значення $G_{max} = 1.0$ усі градієнти масштабуються.

$$g \leftarrow \frac{G_{max}}{|g|_2} \cdot g, \quad (3.29)$$

де g – об'єднаний вектор усіх градієнтів моделі;

$G_{max} = 1.0$ – максимальна допустима норма градієнтів.

Напрямок вектора градієнтів при цьому зберігається, змінюється лише його довжина. Це запобігає ситуації, коли надто великий крок оновлення «викидає» параметри у loss landscape (жахливу область простору втрат).

Під час навчання нейромережі використовується планувальник, який зменшує темп навчання, якщо втрати під час валідації не покращуються протягом 3-х епох поспіль. такому випадку темп навчання множиться на $factor = 0.5$, але не опускається нижче $\eta_{min} = 10^{-4}$. Слід зауважити, що через те що під час навчання буде лише 3 епохи (про це далі), то цього не відбуватиметься у системі. Але теоретично такий функціонал присутній.

Метрики валідації.

Для оцінки моделі використовуються різні метрики. Про них зараз і поговоримо. Обчислюються вони на валідаційній вибірці.

Ассурасу – точність; оцінює загалом кількість правильних відповідей. Зазначена у формулі 3.30.

$$Accuracy = \frac{T + TP}{T + TP + FP + FN}, \quad (3.30)$$

де Ассурасу – частка правильних передбачень серед усіх прикладів.

Приймає значення у проміжку $[0,1]$;

TP – кількість пар, які дійсно належать одному автору і модель правильно їх розпізнає;

T – кількість пар, які належать різним авторам і модель правильно їх розпізнає;

F – хибні ідентифікування, тому що модель помилково вказала одного автора, хоча вони різні;

FN – пропущені ідентифікування, тому що модель помилково вказал різних авторів, хоча фрагменти одного.

Precision – це метрика, що показує з усіх пар, що модель вважає написаним одним автором – яка частка дійсно такими є (написана одним автором). Зазначена у формулі 3.31.

$$\text{Precision} = \frac{T}{T + F}, \quad (3.31)$$

де TP – кількість пар, які дійсно належать одному автору і модель правильно їх розпізнає;

F – хибні ідентифікування, тому що модель помилково вказала одного автора, хоча вони різні.

Recall – це метрика для оцінки, скільки фрагментів з однаковим авторством було знайдено. Дозволяє зрозуміти, скільки однакових фрагментів модель пропускає. Зазначено у формулі 3.32.

$$\text{Recall} = \frac{T}{T + FN}, \quad (3.32)$$

F1-міра – це метрика, що штрафує за дисбаланс між повнотою та упередженістю (тобто між Recall та Precision). Зазначена у формулі 3.33.

$$F_1 = 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \quad (3.33)$$

де F_1 – метрика в проміжку $[0,1]$.

AUC-ROC – це порогонезалежна метрика, яка показує здатність моделі розрізняти позитивні та негативні приклади незалежно від конкретного порогу класифікації. В основі цієї метрики лежить ROC-крива. Вони зазначені у формулах 3.34 та 3.35 відповідно.

$$TPR(\tau) = \frac{T(\tau)}{T(\tau) + N(\tau)}, \quad FPR(\tau) = \frac{FP(\tau)}{FP(\tau) + T(\tau)}, \quad (3.34)$$

де $TPR(\tau)$ – True Positive Rate (повнота) при порозі τ ;

$FPR(\tau)$ – False Positive Rate (частка хибних тривог) при порозі τ ;

$TP(\tau), F(\tau), T(\tau), FN(\tau)$ – компоненти матриці плутанини (confusion matrix) при порозі τ ;

τ – поріг класифікації, $\tau \in [0,1]$.

$$AUC-ROC = \int_0^1 TPR(FPR^{-1}(x)) dx \quad (3.35)$$

Слід зауважити, що вище зазначені формули у загальному вигляді (особливо це стосується AUC-ROC). Для певних практичних цілей можуть бути вирази видозмінені, але статистична суть буде одна й та ж. На практиці AUC-ROC обчислюється за трапецоїдальним правилом [41]. Ця метрика є порогонезалежною, що показує здатність моделі розрізняти позитивні та негативні пари незалежно від класифікаційного порогу. Значення $AUC-ROC = 0.5$ відповідає випадковому вгадуванню (неначе підкидання монетки), а $AUC-ROC = 1.0$ – ідеальному розділенню класів.

3.2 Розробка структурної схеми

Структурна схема системи ідентифікації авторства тексту побудована наступним чином: Уся архітектура розділена на три окремі рівні. Кожен з них виконує певну задану задачу, але пов'язаний з іншими частинами додатку через чітко визначені інтерфейси. Структурна схема зазначена на рисунках 3.2, 3.3, 3.4. Цей підхід до створення архітектури дав змогу провести

розмежування відповідальності кожного із компонентів, спростити подальшу підтримку цього додатку та його розвиток і модифікацію. Оскільки зміни в одній структурі не обов'язково тягнуть за собою переписування функціоналу в інших частинах вихідного коду.

Перший рівень, умовно, можна назвати рівнем збору даних. Він відповідає за отримання, очищення та підготовку текстових даних для навчання, тобто це – препроцесинг. Він розташовується у директорії проекту `src/parsing` і складається з декількох компонентів, які виконуються послідовно. У якості оркестратора тут виступає модуль `Parser`, який координує увесь конвеєр обробки. Спочатку завантажуються дані із `book_sources.py` за допомогою `Downloader`, який є HTTP-клієнтом для взаємодії з веб-інтерфейсом і вміє працювати із зовнішніми джерелами: українською бібліотекою `ukrlib.com.ua` та `Project Gutenberg`. Після завантаження файли передаються до `Cleaner`, який перетворює HTML-сторінки у чистий текстовий формат. Це обумовлено потребою видаляти непотрібну HTML-розмітку, яка є шумом для навчальних даних. Очищені файли зберігаються у файловій системі у директорії `content/raw`. Далі до них застосовується `BookRenamer.py`: компонент, який за допомогою LLM перейменовує файли у канонічний формат за іменем автора. Завдяки цьому легко опрацьовувати складні, нестандартні назви файлів і це дуже допомагає обробляти неструктуровані дані, які зазвичай є на веб-серверах, звідки й завантажується інформація. Останнім компонентом на цьому рівні є `Divider`, який сегментує текст кожного автора на фрагменти визначеної довжини, формуючи структуру даних у вигляді словника, де ключем є ім'я автора, а значенням – список текстових фрагментів. Перший рівень не повинен проводити навчання нейромережі, задавати їй параметри, з нею взаємодіяти. Цим займаються інші рівні і про це ми поговоримо далі. Мета цього рівню – це винятково підготувати навчальні дані, зробити це якісно, швидко, зрозуміло і уніфіковано. Безумовно, певним винятком з цього правила є `Divider`, який розділяє отримані фрагменти, які

потім передаються у неймережу, але цей інтерфейс можна застосувати не тільки для даного додатку, але й використати, наприклад, в якихось інших сферах, де потрібно отримати різні шматки тексту.

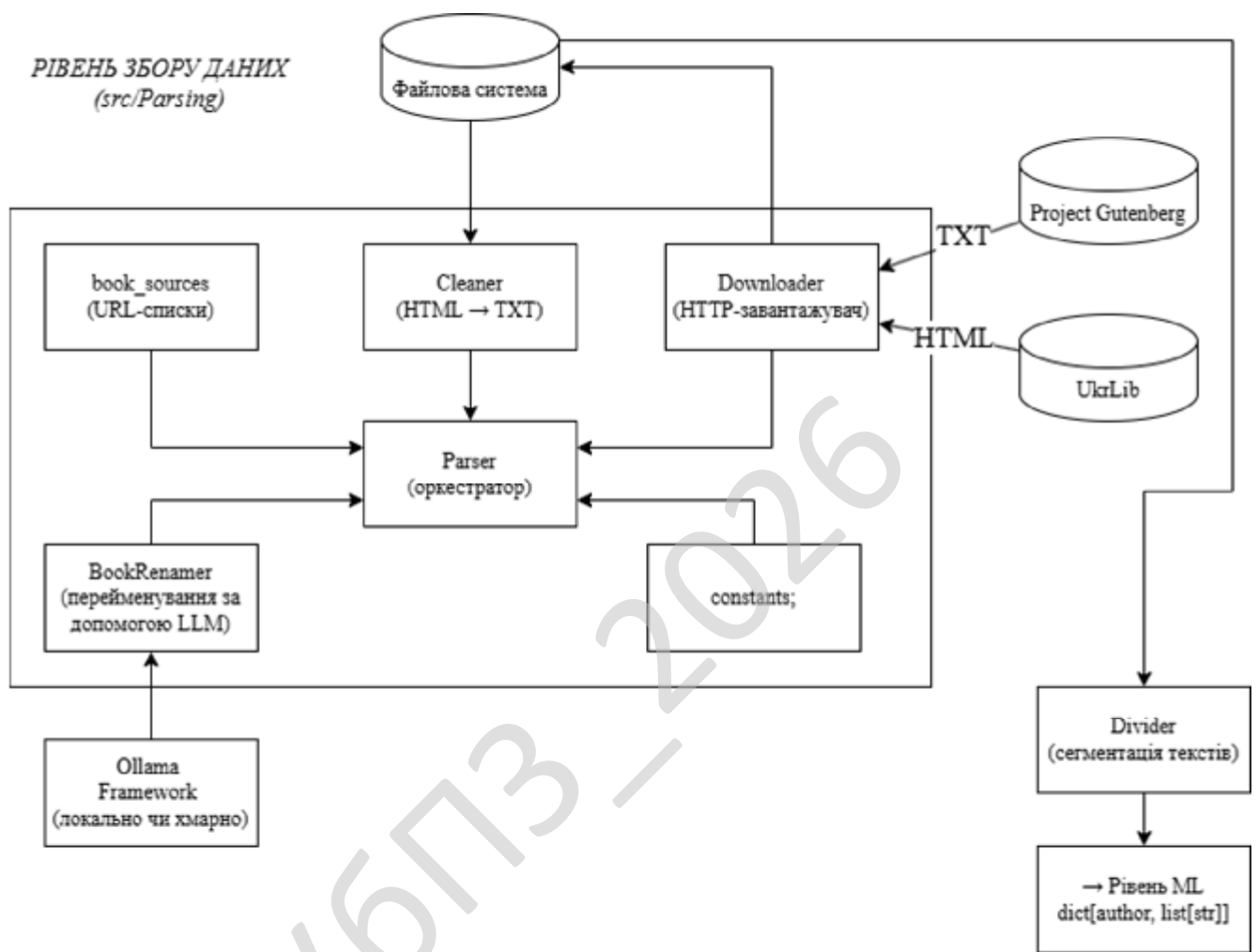


Рисунок 3.2 – Структурна схема (частина перша)

Другий рівень – це ядро системи. Це компонент, який знаходиться у директорії src/IdentificationAI. Він відповідає за навчання нейронної мережі та подальше її використання. Цей рівень складається із багатьох частин і має внутрішні залежності між компонентами.

Першим компонентом можна назвати `nn_config`, який зберігає гіперпараметри моделі і від якого залежать інші компоненти цього рівня. Він отримує дані із файлу формату YAML.

Компонент `vocabulary` виконує задачу токенизацію та стемінгу

українських слів, тобто перетворює текст у послідовність числових індексів, на основі яких потім будуть навчатися embeddings нейронної мережі. Токенізація реалізована з враховуванням специфіки української мови: апострофами, дефісами, складними закінченням слів тощо.

Компонент PairGenerator готує позитивні та негативні пари текстових фрагментів, які в майбутньому будуть подаватись на вхід нейронної мережі під час тренування.

Нейронна мережа SiameseLSTM базується на сіамській архітектурі двонаправленого LSTM та обробляє два тексти паралельно через спільний encoder. І лише потім порівнює отримані вектори за допомогою класифікатору.

Компонент AuthorshipMetrics акумулює метрики оцінки: точність, повноту, F1-score, AUC-ROC під час тренування нейронної моделі. Слід зауважити, що тут потрібно відокремити компонент і клас. Останній не містить у собі усі ці метрики оцінки. Але загалом визначення ефективності тренування параметрів нейронної мережі винесено як окремий логічно виділений компонент.

У процесі реалізації архітектури програмного забезпечення засобами мови Python було зроблено ряд спрощувальних припущень, продиктованих практичними міркуваннями розробки. На це вплинула і практика об'єктно-орієнтованого програмування, функціонального програмування; патерни проектування та потреба у прискоренні обчислювальних операцій. Тому важливим є зберігання балансу між «чистотою коду» та нагальними потребами системи.

Компонент AuthorVerifier орієнтований на роботу з попередньо навченою нейронною моделлю: він завантажує збережений checkpoint, конфігурує модель відповідно до визначених гіперпараметрів і надає функціональність для порівняльного аналізу двох текстових фрагментів із метою встановлення їх належності одному автору.

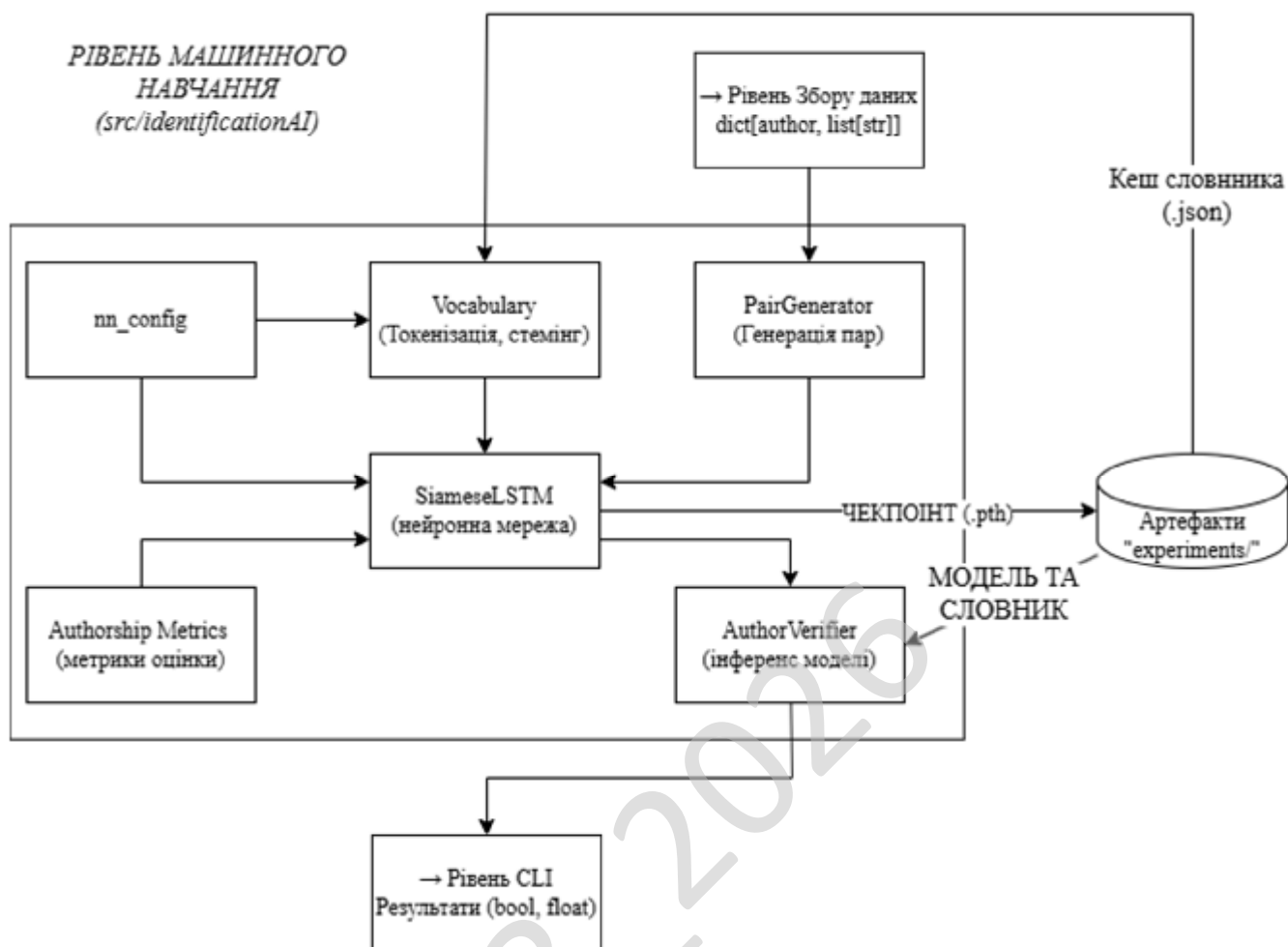


Рисунок 3.3 – Структурна схема (частина друга)

Третій рівень – це рівень інтерфейсу користувача. Він дозволяє комунікувати із вже навченою нейронною мережею без змін у вихідному коді. Особливо у цьому рівні виділяється файл cli.py: Там реалізована логіка парсингу аргументів, повідомлення користувача, вивід результатів, формування звіту в форматах XML, JSON, TXT. Він також агрегує результати порівняння. Якщо тексти занадто великі, вони розбиваються на шматки по N токенів і аналізуються (у контексті реалізації найбільш оптимальним є значення у тисячу токенів). Для агрегування даних можна застосувати середнє арифметичне, середнє геометричне, медіанне та гармонічне середнє – усі вони є окремими випадками середнього по Колмогорову. У залежності від вхідних даних (науковий, діловий текст та ін.) можна підлаштовувати систему.

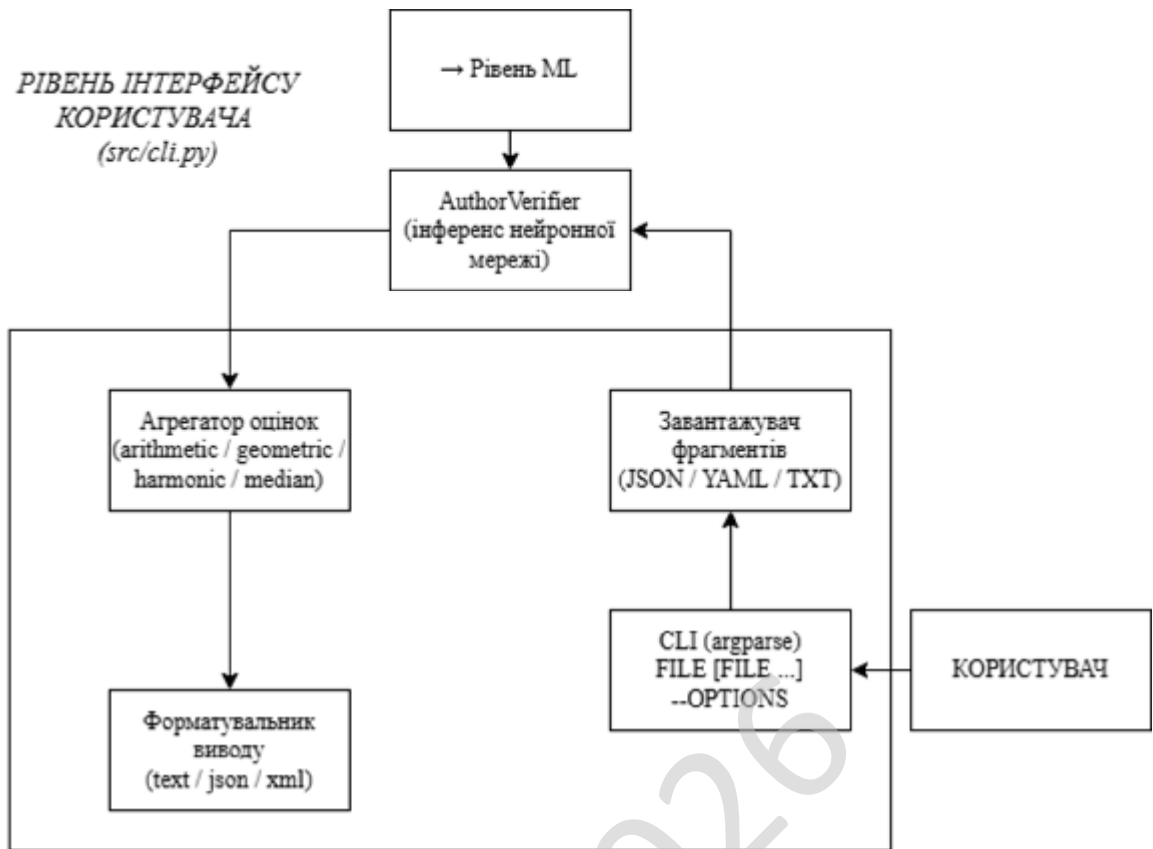


Рисунок 3.4 – Структурна схема (частина третя)

3.3 Розробка функціональної схеми

Спочатку слід розглянути функціональну схему програми ідентифікації авторства тексту. Вона складається із трьох частин, вони наведені нижче у кожному абзаці. Рішення розбити на три частини обумовлено тим, що на одному зображенні не можна умістити усі компоненти. Навіть якщо така можливість була б, то краще для візуального сприйняття зробити декілька рисунків. Функціональна схема знаходиться на рисунках 3.5, 3.6, 3.7.

Перша функціональна частина (Ф1) відповідає за препроцесинг текстових даних – від завантаження сирих HTML-сторінок до формування структурованого корпусу, придатного для навчання нейронної мережі. Вхідними даними слугують URL-посилання, що зберігаються у конфігураційному файлі `book_sources.py` і вказують на джерела текстів

конкретних авторів.

Стратегія розбиття корпусу є дворівневою. Основний тренувальний набір містить підмножини для навчання та внутрішньої валідації, що дає змогу відстежувати динаміку навчання і запобігати перенавчанню. Водночас формується окремий валідаційний набір, до якого включаються виключно ті автори, що у жодному вигляді не представлені у тренувальному корпусі. Така організація даних суттєво підвищує об'єктивність оцінки моделі: за відсутності цього обмеження мережа могла б підлаштовуватись під ідіостиль конкретних авторів у процесі навчання, а оцінка на внутрішній валідаційній підмножині не відображала б реальної здатності до ідентифікації незнайомих авторів.

Модуль Downloader виконує HTTP-запити до вказаних джерел і зберігає отримані дані порціями по 10 МБ. Завантажені URL кешуються у файл `download_cache.json`, що дозволяє при повторному запуску пайплайну пропускати вже отримані ресурси.

Наступним кроком модуль Cleaner виконує парсинг завантажених HTML-файлів засобами бібліотеки BeautifulSoup4. У процесі парсингу видаляються теги розмітки, навігаційні елементи, службові блоки та метадані сторінок, тобто усі складові, що не є безпосередньою частиною авторського тексту. Результатом є чистий текстовий файл, придатний до подальшої обробки.

BookRenamer використовує велику мовну модель через інтерфейс Ollama для аналізу та стандартизації імен файлів. Вихідні назви файлів, як правило, не відповідають єдиному стандарту: вони можуть містити аббревіатури, транслітерацію, порядкові номери у різних форматах тощо. BookRenamer приводить їх до уніфікованого формату `прізвище-ім'я-номер.txt`, що є необхідною умовою коректної роботи подальших модулів. На цьому етапі передбачена можливість вибору між локально розгорнутою моделлю та хмарними обчисленнями через сервіс Ollama Cloud.

Завершальним кроком є модуль Divider. Він групує усі тексти за авторською приналежністю, об'єднує їх у єдиний суцільний документ і рівномірно поділяє на N фрагментів. Підсумковим результатом роботи Ф1 є корпус текстів у вигляді словника, де ключем виступає ідентифікатор автора, а значенням – список відповідних текстових фрагментів.

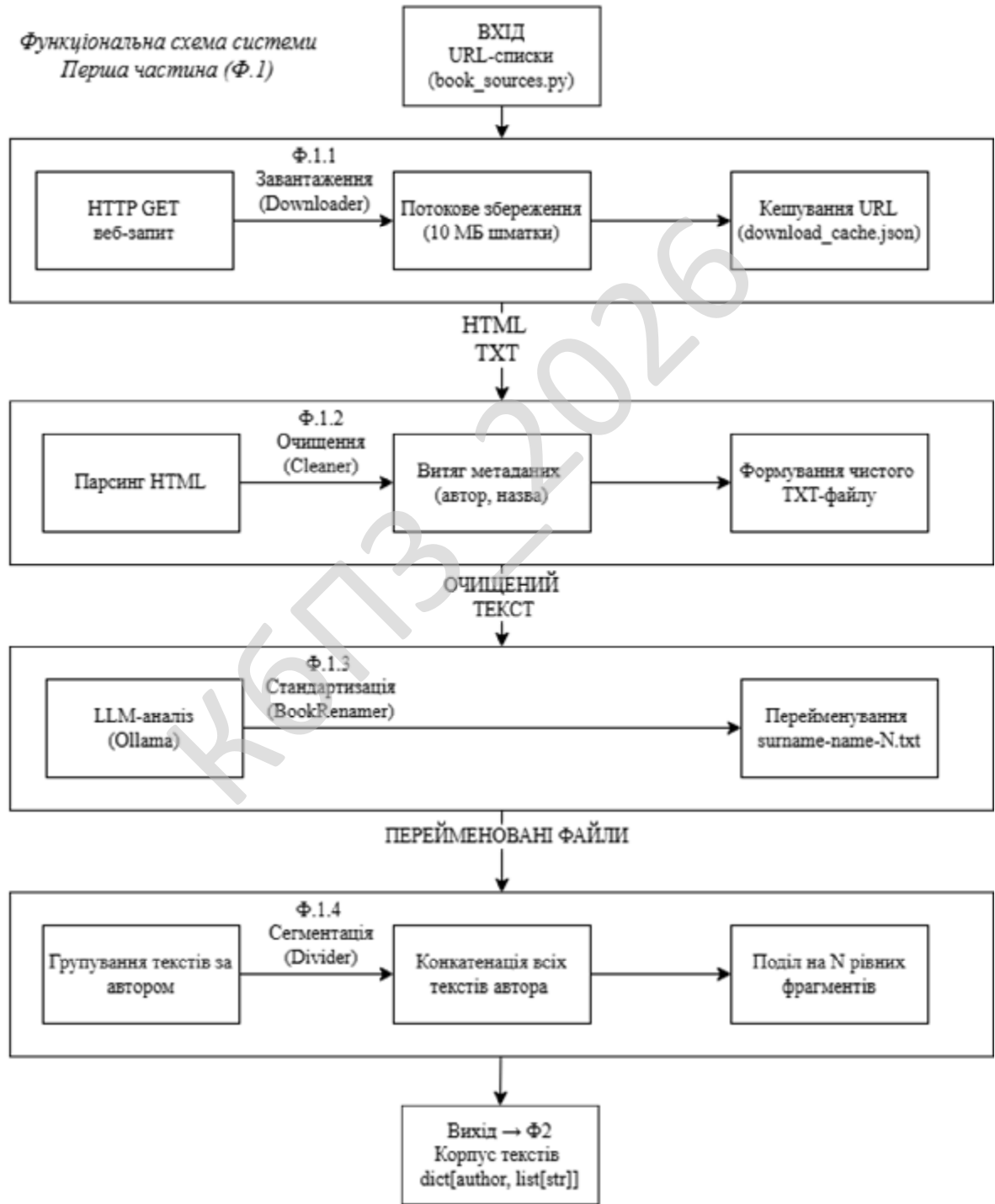


Рисунок 3.5 – Функціональна схема (частина перша)

Друга частина (Ф2) функціональної схеми формалізує процес навчання нейронної мережі. На вхід подається корпус текстів, який береться безпосередньо із даних з попереднього етапу, а також конфігурація з файлу формату YAML, де задано усі необхідні параметри.

Спочатку відбувається створення словника (vocabulary): тексти токенізуються за допомогою регулярних виразів, застосовується стемінг. Потім фільтруються слова і відкидаються ті, які мають частоту менше заданого натурального числа (емпірично виявлено, що оптимальне значення від двох до чотирьох).

Потім відбувається кешування наступним шляхом: відстежується стан директорії (назви файлів та час їх зміни) і ці дані хешуються за допомогою алгоритму MD5. Завдяки цьому усувається потреба кожен раз проводити створення словника. Така дія відбувається тільки у тому випадку, коли змінюється стан директорії у файловій системі.

Наступним кроком генератор пар формує навчальний та валідаційний набір. Даними при такому підході наступні: позитивні пари – це фрагменти одного автора (мітка 1), та негативні – це фрагменти різних авторів (мітка 0).

Відтепер можна приступити до навчання нейронної мережі. Спочатку відбувається прямий прохід через LSTM-мережу, потім обчислюється функція втрат, відбувається backpropagation (метод зворотного поширення помилки), опісля обрізка градієнтів із нормою менше 1.0, оновлюється ваги оптимізатором AdamW та коригується швидкість навчання планувальником ReduceLROnPlateau.

Для кожної епохи є валідація, де обчислюються метрики: Accuracy, Precision, Recall, F1 та AUC-ROC. На виході формуються три файли: checkpoint, кеш словника та звіт про навчання у форматі XML.

Такий підхід забезпечує розділення відповідальності між компонентами, запобігає зрощенню функціоналу різних класів, а також коректно відображає предметну область.

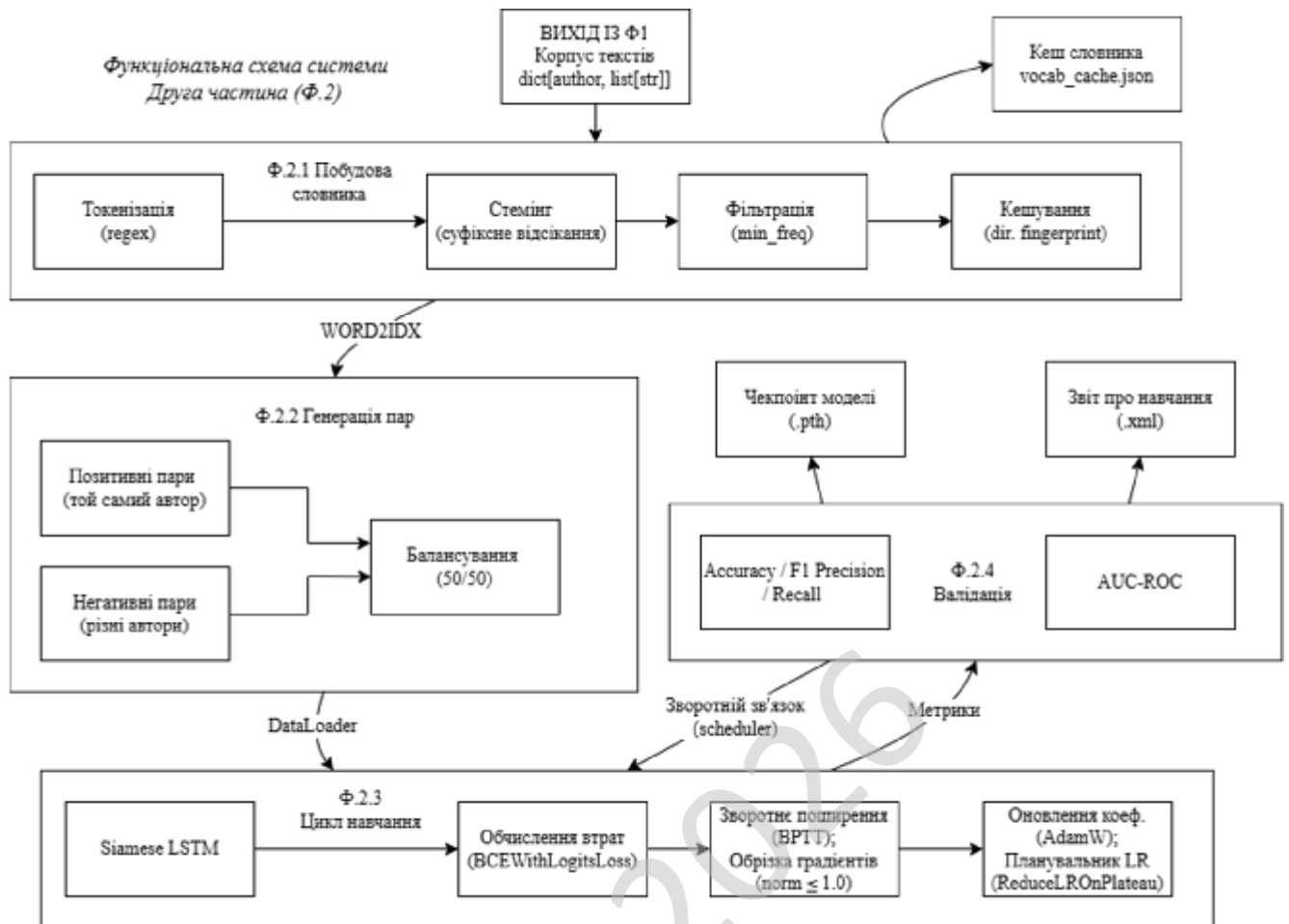


Рисунок 3.6 – Функціональна схема (частина друга)

Третя частина (Ф3) відноситься до ідентифікації авторства. Це в основному стосується CLI-утиліти для аналізу текстів. На вхід подаються фрагменти у форматах JSON, YAML або TXT. Завантажуються checkpoint моделі, кеш словника та конфігурація. Проводиться попередня обробка шляхом завантаження фрагментів із файлів у вищезазначених форматах. Вони поділяються на шматки (chunks) не більше 1000 токенів. Надалі модуль AuthorVerifier кодує текст, тобто проводить векторизацію, отримує результат від сіамської мережі та проводить його через сигмоїдну функцію. Надалі ми проводимо декартов добуток усіх шматків між собою, після чого оцінки від нейромережі агрегуються одним із методів: арифметичне середнє, геометричне середнє, гармонічне середнє або медіана. Фінальне рішення приймається на базі порогового значення 0.65. Якщо агрегована ймовірність

перевищує цей поріг, ми вважаємо, що два фрагменти належать одному автору; якщо ж ні, вони написані різними авторами. Результати можуть бути виведені у форматах TXT, JSON або XML. Вони можуть бути або записані у файлову систему, або просто виведені у якийсь потік (наприклад, до терміналу).

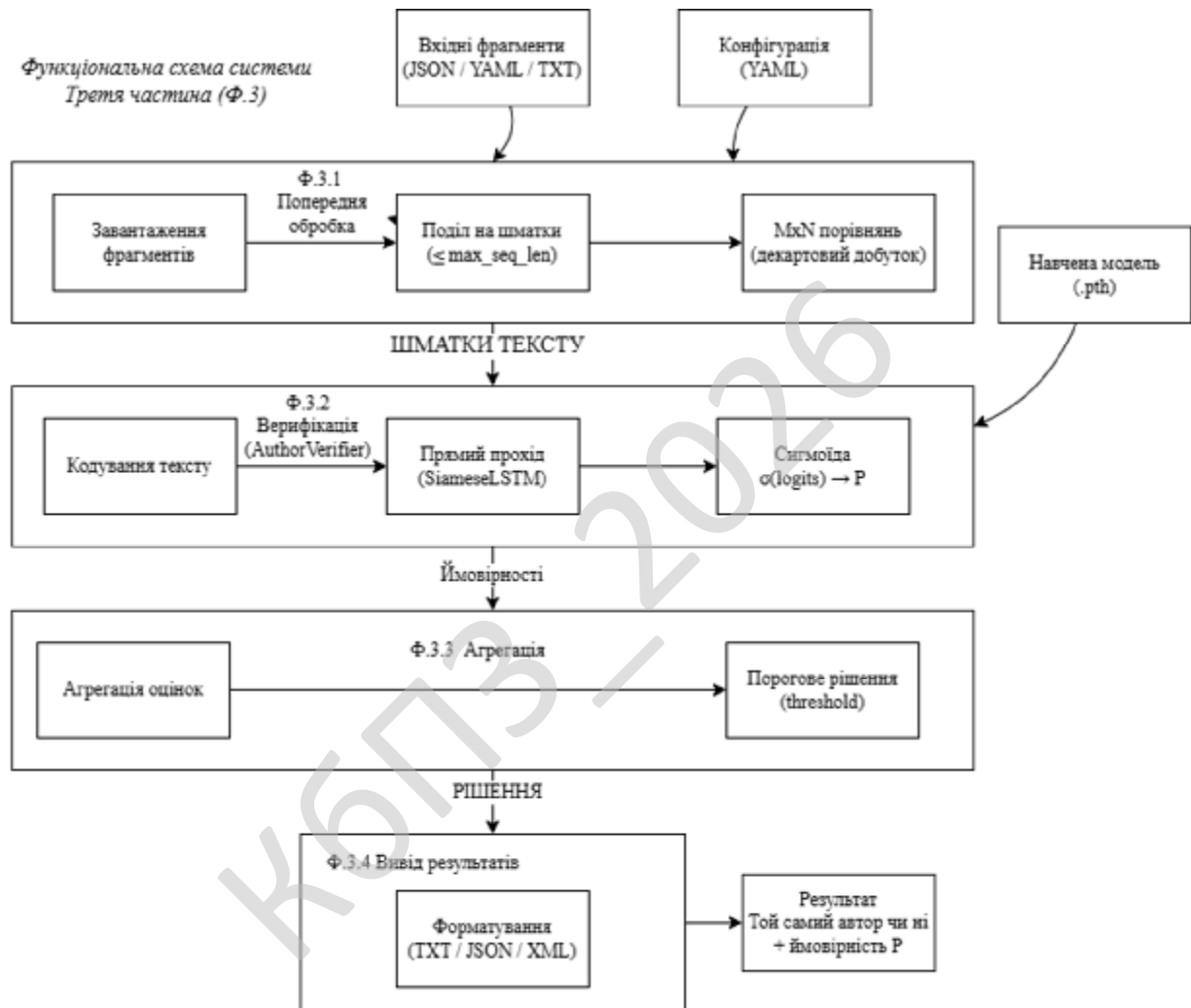


Рисунок 3.7 – Функціональна схема (частина третя)

3.4 Розробка діаграми процесів

Діаграма процесів дозволяє відобразити життєвий цикл розробленої системи. Це стосується як збору корпусу текстів, так і безпосередньо ідентифікації авторства тексту на етапі інференсу. Діаграма відображає

систему як єдиний конвеєр – від «сирих» даних до бінарного рішення. Принципова особливість – це те, що програма побудована як ланцюг автономних етапів (фаз), кожен з яких виконує рівно одну функцію і передає свій результат наступному. [42]

Перші два блоки, а саме: завантаження текстів та очищення текстів, відповідають за перехід від неструктурованих зовнішніх джерел до уніфікованого внутрішнього представлення. На цьому етапі система абстрагується від конкретного формату джерела і зводить усе до єдиного формату – чистого тексту для подальшої обробки. Це критично важливо особливо тому що якість вхідних даних визначає усю майбутню роботу системи. Вона зазначена на рисунку 3.8.

Від блоку очищення текстів відходять дві паралельні гілки, а саме: побудова словника та формування пар текстів. Це свідомий вибір під час проектування: словник і генерація пар є двома різними компонентами, але кожен із них будується на основі очищених текстів. Словник визначає кодування вхідних даних числами, тобто проводить бієктивне відображення між символами та натуральними числами, а генератор пар визначає, які тексти використовувати під час навчання.

Така паралельність підкреслює модульність системи, тому що зміна стратегії токенизації (наприклад, перехід від суфіксного стемінгу до лематизації) не вимагає зміни логіки формування пар, водночас, і навпаки – зміна пропорції позитивних та негативних пар не впливає на побудову словника.

Ядром системи є зв'язка блоків навчання моделі та валідації. Цей цикл – це загальний принцип машинного навчання, який корінним чином не змінюється від предметної області. Збереження чекпоінту дозволяє зберегти найкращий стан. Такий підхід гарантує, що перенавчання буде мінімально впливати на кінцеву модель.



Рисунок 3.8 – Діаграма процесів

Нижня частина діаграми описує процес використання моделі. Під час інференсу система працює із вже навченою моделлю та конкретною парою текстів для отримання конкретної відповіді. За це відповідають блоки порівняння текстів та агрегація результатів.

Діаграма в цілому демонструє, що програма побудована за принципом звуження інформації: від широких сирих даних – до очищених корпусів, від очищених корпусів – до визначених обрізаних фрагментів, від фрагментів – до числових індексів та векторних представлень, від векторів – до скалярного представлення і в кінці-кінців до булевого рішення – один автор чи різні.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Не усі частини системи ідентифікації авторства тексту можуть бути описані діаграм із минулих розділів або ж за допомогою текстового опису. Для більш детального розгляду потрібно навести блок-схеми. Блок-схема основної програми наведена на рисунку 4.1.

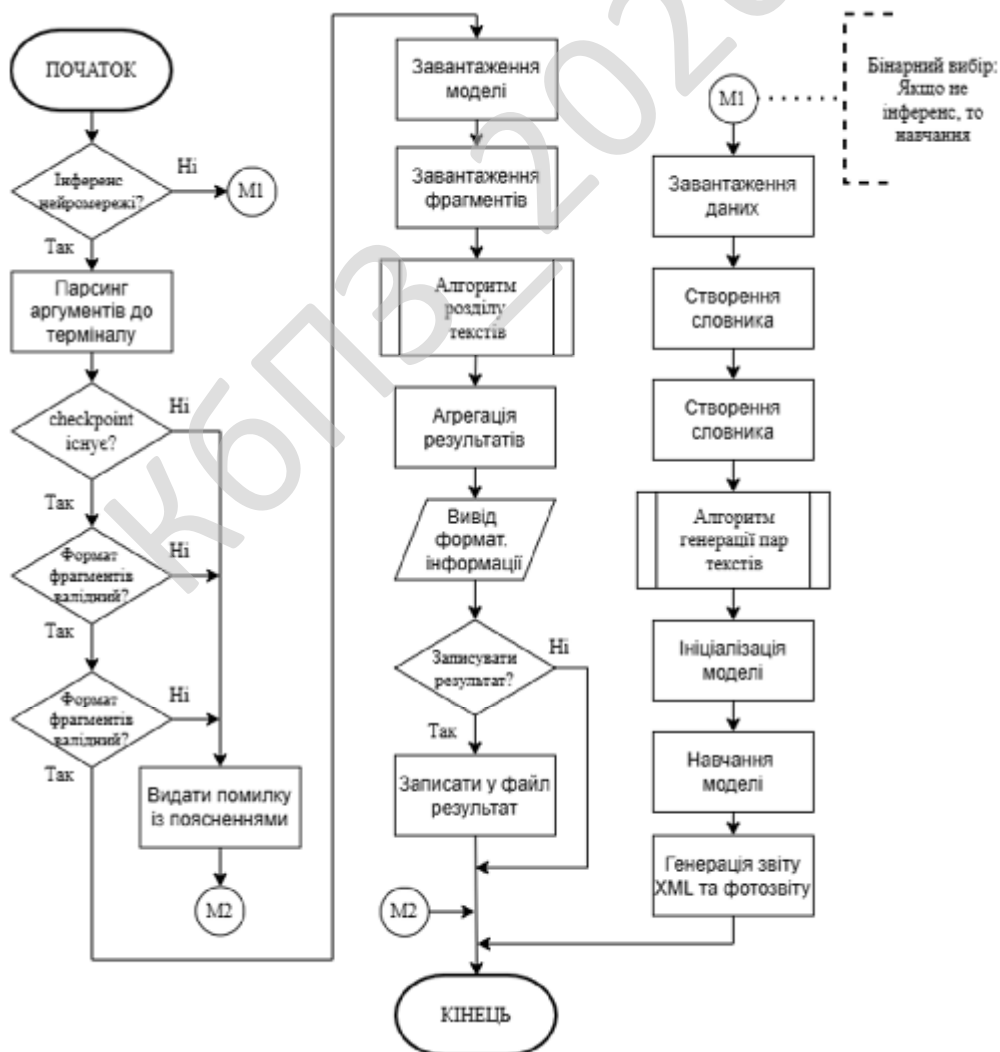


Рисунок 4.1 – Блок-схема основної програми

Спочатку слід розглянути два важливих алгоритма, що необхідні для системи, а саме: розділ текстів та генерація пар текстів. Вони наведені на рисунках 4.2. та 4.3. відповідно.

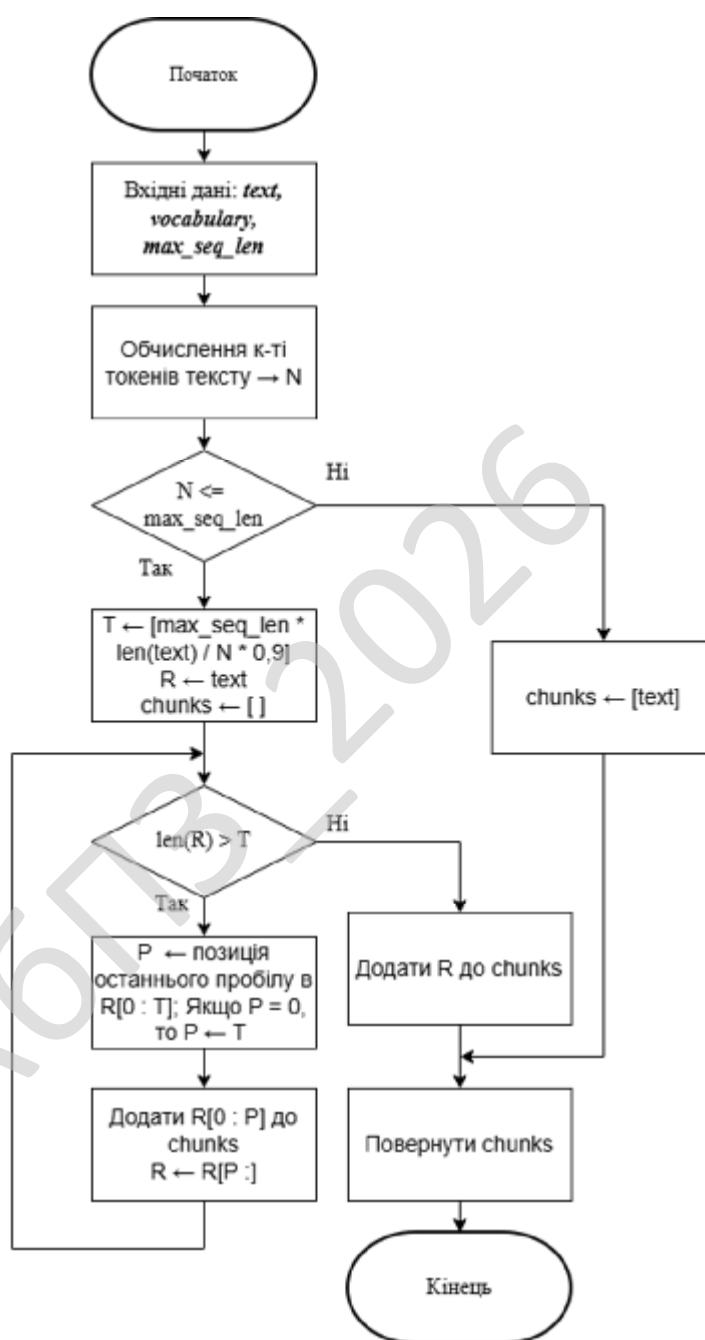


Рисунок 4.2 – Алгоритм розділу текстів

Алгоритм розділу текстів демонструє підхід, коли вхідний текст перебільшує значення максимальної кількості символів. І у такому випадку

потрібно його «дробити» на менші шматки. Але згодом потрібно формувати пари текстів (так і пари від одного автора, так і від різних). І алгоритм цього зазначено на рисунку 4.3.



Рисунок 4.3 – Алгоритм генерації пар текстів

Серед експериментальних даних слід виділити метрики, графічні візуалізації та вивід лог-інформації у термінал.

Рисунок із графічною візуалізацією навчання зазначений у 4.4. На ньому зображено точність на валідаційному датасеті. На ньому зображено інформацію для 3-х епох процесу навчання (їх усього було проведено 3). Така кількість епох була обумовлена невеликим датасетом, який вдалося зібрати. Незважаючи на те, що він містив десятки творів, все одно нейромережа такого розміру потребує більших навчальних даних. Але результати в будь-якому випадку виявились вражаючими: точність сягає майже 94% на валідаційному датасеті. При цьому датасет містить у собі невідомі уривки текстів, тобто ці фрагменти вона не бачила під час навчання. Але є важливе зауваження: хоча вона і не бачила безпосередньо уривки, вона все одно навчалась на стилі цих письменників. Тобто, можливо, нейромережа і не запам'ятовує тексти, але все ще є ймовірність того, що вона «підладжується» під конкретних авторів. І якщо дати якихось невідомих авторів, усе може істотно погіршитися. Для того щоб це зрозуміти було зроблено інший набір даних, що містив у собі лише авторів, твори яких у жодному виді не попадалися під час навчання або ж валідації (назвемо її первинною). Отож потрібно розглянути результати моделі на вторинному датасеті для валідації.

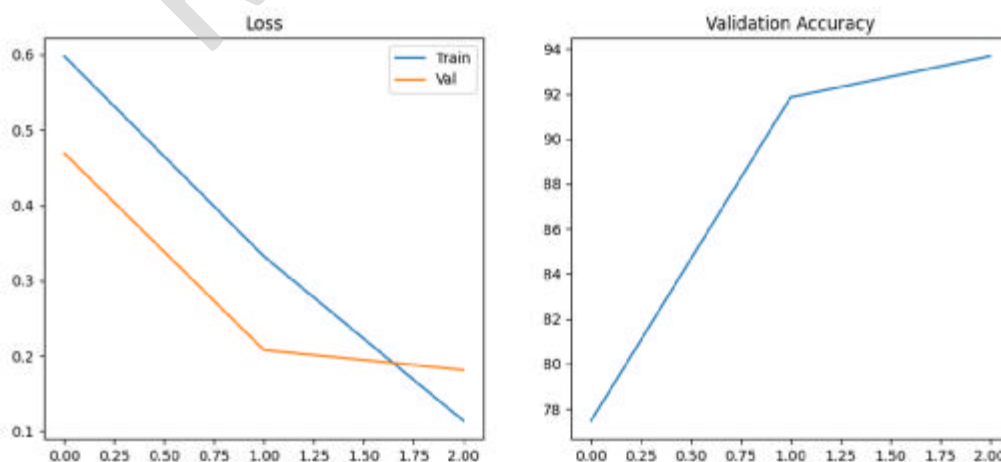


Рисунок 4.4 – Графічна візуалізація для 3-х епох навчання

Інформаційний вивід під час навчання дає повну картину того, що відбувається. Він вказує пристрій, що використовується для обчислень, початкове зерно для генерації пар (оскільки у вихідному коді є випадкові числа), а також виводить метрики у термінал. Термінальний вивід зазначений нижче.

```
20:30:07 [INFO] __main__: Device: cuda | Seed: 2026
20:30:07 [INFO] __main__: --- 1. Loading Data ---
20:30:07 [INFO] parsing.Divider: Start of reading and grouping of
files
20:30:08 [INFO] parsing.Divider: Handling 22 authors...
20:30:08 [INFO] __main__: Total authors loaded: 22
20:30:08 [INFO] __main__: Authors in Train: 22 | Authors in Val: 22
20:30:08 [INFO] __main__: Vocabulary cache is up-to-date: loading from
B:\Projects\Current\Python\AuthorIdentification\experiments\vocab_cache.json
20:30:08 [INFO] identificationAI.vocabulary: Vocabulary loaded from
B:\Projects\Current\Python\AuthorIdentification\experiments\vocab_cache.json
(76060 tokens)
20:30:08 [INFO] __main__: --- Generating Pairs ---
20:30:08 [INFO] identificationAI.pair_generator: Generating pairs for
22 authors...
20:30:08 [INFO] identificationAI.pair_generator: Generating pairs for
22 authors...
20:30:08 [INFO] __main__: Train pairs: 11000 | Val pairs: 6600
=====
20:30:11 [INFO] __main__: --- Starting Training ---
20:32:31 [INFO] __main__: Epoch 1/3 | Loss: 0.5868 | Val Loss: 0.4813
| Acc: 76.97% | F1: 0.7986 | Prec: 0.7095 | Rec: 0.9133 | AUC: 0.8829 | LR:
5.00e-04 | GradNorm: 1.5047
20:32:31 [INFO] __main__: Checkpoint saved (val_loss=0.4813) →
B:\Projects\Current\Python\AuthorIdentification\experiments\siamese_full_pack
age.pth
20:34:51 [INFO] __main__: Epoch 2/3 | Loss: 0.3356 | Val Loss: 0.2421
| Acc: 90.56% | F1: 0.9053 | Prec: 0.9082 | Rec: 0.9024 | AUC: 0.9660 | LR:
5.00e-04 | GradNorm: 1.6044
20:34:51 [INFO] __main__: Checkpoint saved (val_loss=0.2421) →
B:\Projects\Current\Python\AuthorIdentification\experiments\siamese_full_pack
age.pth
20:37:10 [INFO] __main__: Epoch 3/3 | Loss: 0.1832 | Val Loss: 0.3913
| Acc: 86.20% | F1: 0.8525 | Prec: 0.9155 | Rec: 0.7976 | AUC: 0.9501 | LR:
5.00e-04 | GradNorm: 1.2399 ...
```

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

69

Слід зауважити, що частина даних була обрізана, щоб не демонструвати найменші аспекти (котрі зараз не грають великої ролі у вищезазначеному абзаці). Більш детально функціонал зазначений у вихідному коді, де можна побачити усі данні, які можуть виводитись, а також звідки вони беруться.

Приклади термінальних команд наведені нижче. Вони дозволяють проводити попарне порівняння двох файлів, порівнювати усі файли у директорії і виводити звіт у XML-файл, верифікувати із заданим порогом і виводом JSON у stdout. Вивід від першої команди зазначений на рисунку 4.5.

```
python .\src\main.py `
    content\raw\ukr_data\test_data\fedoriv-roman-mykolayovych-1.txt `
    content\raw\ukr_data\test_data\havrosh-oleksandr-diulovych-1.txt `
    --output text --progress
...
python .\src\main.py `
    content\raw\ukr_data\test_data\*.txt `
    --output xml -o results\test_output.xml `
    --aggregation harmonic --progress
python .\src\main.py `
    content\raw\ukr_data\test_data\kosach-iuriy-mykolayovych-1.txt `
    content\raw\ukr_data\test_data\zahrebelnyy-pavlo-arkhypovych-1.txt `
    content\raw\ukr_data\test_data\vynnychenko-volodymyr-kyrylovych-
1.txt `
...
    --threshold 0.70 --aggregation median --output json
```

Проте слід зауважити, що це не усі можливі варіанти термінальних команд. В залежності від потреб, наявних файлів та обчислювальних ресурсів це може змінюватись. Також варто розглянути приклад звіту про навчання нейронної мережі.

```
<?xml version='1.0' encoding='utf-8'?>
<training_report status="completed">
  <meta>
    <training_duration_seconds>418.4</training_duration_seconds>
    <seed>2026</seed>
    <device>cuda</device>
  </meta>
```

```

<model>
  <total_params>23150081</total_params>
  <trainable_params>23150081</trainable_params>
</model>
<config>
  <model>
    <embed_dim>256</embed_dim>
    <hidden_dim>384</hidden_dim>
    <classifier_h1>512</classifier_h1>
    <classifier_h2>256</classifier_h2>
    <dropout>0.35</dropout>
    <embed_dropout>0.2</embed_dropout>
  </model>
  <training>
    <lr>0.0005</lr>
    <epochs>3</epochs>
    <batch_size>32</batch_size>
    <scheduler_factor>0.5</scheduler_factor>
    <scheduler_patience>3</scheduler_patience>
    <scheduler_min_lr>1e-06</scheduler_min_lr>
    <grad_clip_max_norm>1.0</grad_clip_max_norm>
    <weight_decay>0.0001</weight_decay>
  </training>
  <data>
    <max_seq_len>1000</max_seq_len>
    <min_freq>2</min_freq>
    <train_samples_per_author>250</train_samples_per_author>
    <val_samples_per_author>150</val_samples_per_author>
    <divider_target_count>300</divider_target_count>
  </data>
</config>
<epochs>
  <epoch number="1">
    <train_loss>0.586840</train_loss>
    <val_loss>0.481306</val_loss>
    <val_acc>76.9697</val_acc>
    <val_f1>0.7986</val_f1>
    <val_precision>0.7095</val_precision>
    <val_recall>0.9133</val_recall>
    <val_auc_roc>0.8829</val_auc_roc>
    <lr>0.00050000</lr>
    <avg_grad_norm>1.5047</avg_grad_norm>
  </epoch number="1">

```

```

</epoch>
<epoch number="2">
  <train_loss>0.335636</train_loss>
  <val_loss>0.242065</val_loss>
  <val_acc>90.5606</val_acc>
  <val_f1>0.9053</val_f1>
  <val_precision>0.9082</val_precision>
  <val_recall>0.9024</val_recall>
  <val_auc_roc>0.9660</val_auc_roc>
  <lr>0.00050000</lr>
  <avg_grad_norm>1.6044</avg_grad_norm>
</epoch>
<epoch number="3">
  <train_loss>0.183200</train_loss>
  <val_loss>0.391310</val_loss>
  <val_acc>86.1970</val_acc>
  <val_f1>0.8525</val_f1>
  <val_precision>0.9155</val_precision>
  <val_recall>0.7976</val_recall>
  <val_auc_roc>0.9501</val_auc_roc>
  <lr>0.00050000</lr>
  <avg_grad_norm>1.2399</avg_grad_norm>
</epoch>
</epochs>
</training_report>
...

21:41:49 [INFO] __main__: Device: cuda | Seed: 2022
21:41:49 [INFO] __main__: --- 1. Loading Data ---
21:41:49 [INFO] parsing.Divider: Start of reading and grouping of files
21:41:49 [INFO] parsing.Divider: Handling 22 authors...
21:41:49 [INFO] __main__: Total authors loaded: 22
21:41:49 [INFO] __main__: Authors in Train: 22 | Authors in Val: 22
21:41:49 [INFO] __main__: Vocabulary cache is up-to-date, loading from
B:\Projects\Current\Python\AuthorIdentification\experiments\vocab_cache.json
21:41:49 [INFO] identificationAI.vocabulary: Vocabulary loaded from
B:\Projects\Current\Python\AuthorIdentification\experiments\vocab_cache.json
(76060 tokens)
21:41:49 [INFO] __main__: --- Generating Pairs ---
21:41:49 [INFO] identificationAI.pair_generator: Generating pairs for
22 authors...
21:41:49 [INFO] identificationAI.pair_generator: Generating pairs for
22 authors...

```

```

21:41:49 [INFO] __main__: Train pairs: 11000 | Val pairs: 6600
21:41:50 [INFO] __main__:
=====
21:41:50 [INFO] __main__: MODEL PARAMETER SUMMARY
21:41:50 [INFO] __main__: Total parameters: 23150081
21:41:50 [INFO] __main__: Trainable parameters: 23150081
21:41:50 [INFO] __main__: Non-trainable: 0
21:41:50 [INFO] __main__: -----
-----
21:41:50 [INFO] __main__: embedding.weight
[76060, 256] (19471360)
21:41:50 [INFO] __main__: lstm.weight_ih_l0
[1536, 256] (393216)
21:41:50 [INFO] __main__: lstm.weight_hh_l0
[1536, 384] (589824)
21:41:50 [INFO] __main__: lstm.bias_ih_l0
[1536] (1536)
21:41:50 [INFO] __main__: lstm.bias_hh_l0
[1536] (1536)
21:41:50 [INFO] __main__: lstm.weight_ih_l0_reverse
[1536, 256] (393216)
21:41:50 [INFO] __main__: lstm.weight_hh_l0_reverse
[1536, 384] (589824)
21:41:50 [INFO] __main__: lstm.bias_ih_l0_reverse
[1536] (1536)
21:41:50 [INFO] __main__: lstm.bias_hh_l0_reverse
[1536] (1536)
21:41:50 [INFO] __main__: classifier.0.weight
[512, 3072] (1572864)
21:41:50 [INFO] __main__: classifier.0.bias [512]
(512)
21:41:50 [INFO] __main__: classifier.1.weight
[512] (512)
21:41:50 [INFO] __main__: classifier.1.bias [512]
(512)
21:41:50 [INFO] __main__: classifier.4.weight
[256, 512] (131072)
21:41:50 [INFO] __main__: classifier.4.bias [256]
(256)
21:41:50 [INFO] __main__: classifier.5.weight
[256] (256)

```

```

21:41:50 [INFO] __main__: classifier.5.bias [256]
(256)
21:41:50 [INFO] __main__: classifier.8.weight [1,
256] (256)
21:41:50 [INFO] __main__: classifier.8.bias [1]
(1)
21:41:50 [INFO] __main__:
=====
21:41:52 [INFO] __main__: --- Starting Training ---
21:44:06 [INFO] __main__: Epoch 1/5 | Loss: 0.5734 | Val Loss: 0.4324 |
Acc: 79.97% | F1: 0.7994 | Prec: 0.8006 | Rec: 0.7982 | AUC: 0.8832 | LR: 5.00e-
04 | GradNorm: 1.6014
21:44:06 [INFO] __main__: Checkpoint saved (val_loss=0.4324) ->
B:\Projects\Current\Python\AuthorIdentification\experiments\siamese_full_pack
age.pth
21:46:19 [INFO] __main__: Epoch 2/5 | Loss: 0.3249 | Val Loss: 0.3663 |
Acc: 84.35% | F1: 0.8339 | Prec: 0.8881 | Rec: 0.7861 | AUC: 0.9331 | LR: 5.00e-
04 | GradNorm: 1.3585
21:46:20 [INFO] __main__: Checkpoint saved (val_loss=0.3663) ->
B:\Projects\Current\Python\AuthorIdentification\experiments\siamese_full_pack
age.pth
21:48:33 [INFO] __main__: Epoch 3/5 | Loss: 0.1655 | Val Loss: 0.3401 |
Acc: 86.74% | F1: 0.8570 | Prec: 0.9301 | Rec: 0.7945 | AUC: 0.9577 | LR: 5.00e-
04 | GradNorm: 1.1799
21:48:34 [INFO] __main__: Checkpoint saved (val_loss=0.3401) ->
B:\Projects\Current\Python\AuthorIdentification\experiments\siamese_full_pack
age.pth
21:50:48 [INFO] __main__: Epoch 4/5 | Loss: 0.0802 | Val Loss: 0.3973 |
Acc: 88.12% | F1: 0.8781 | Prec: 0.9014 | Rec: 0.8561 | AUC: 0.9564 | LR: 5.00e-
04 | GradNorm: 1.0200
21:53:00 [INFO] __main__: Epoch 5/5 | Loss: 0.0491 | Val Loss: 0.3727 |
Acc: 90.20% | F1: 0.9002 | Prec: 0.9167 | Rec: 0.8842 | AUC: 0.9681 | LR: 5.00e-
04 | GradNorm: 0.8123
21:53:00 [INFO] __main__: Training finished. Best val_loss: 0.3401 |
Duration: 668.3s
21:53:00 [INFO] __main__: Training report saved ->
B:\Projects\Current\Python\AuthorIdentification\experiments\training_report.x
ml
21:53:00 [INFO] __main__: Plot saved ->
B:\Projects\Current\Python\AuthorIdentification\src\identificationAI\training
_result.png
PS B:\Projects\Current\Python\AuthorIdentification>

```

Рим	Анк	№ доквм.	Підпис	Печат

БКРБ-122.26.0025.00.00.ПЗ

Арк.

74

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Threshold: 0.65 | Aggregation: arithmetic

--- Comparison 1/1 ---
A: "fedoriv-roman-nykolayovych-1" (41747 tokens + 47 chunks)
B: "havrosh-oleksandr-diulovych-1" (10098 tokens + 13 chunks)
Pairwise: [0.6176, 0.4698, 0.5956, 0.3427, 0.1733, 0.3552, 0.1888, 0.28, 0.4152, 0.3214, 0.
95, 0.0511, 0.0889, 0.1181, 0.1106, 0.0962, 0.056, 0.0114, 0.1192, 0.1067, 0.1291, 0.0761, 0.
099, 0.0647, 0.0792, 0.1075, 0.0682, 0.0275, 0.0341, 0.0257, 0.0371, 0.0479, 0.0415, 0.0353,
0.0269, 0.0313, 0.0445, 0.0426, 0.0377, 0.0271, 0.0104, 0.5653, 0.723, 0.6861, 0.5711, 0.2167,
0.1775, 0.1291, 0.19, 0.1088, 0.0613, 0.1119, 0.04, 0.0812, 0.1519, 0.1164, 0.0948, 0.059, 0.
0699, 0.0899, 0.0884, 0.1026, 0.0652, 0.0127, 0.4787, 0.3355, 0.4443, 0.2666, 0.1063, 0.146,
0.189, 0.3109, 0.1494, 0.097, 0.2206, 0.0571, 0.1343, 0.2106, 0.1983, 0.1655, 0.125, 0.0118,
0.1286, 0.1064, 0.0973, 0.0665, 0.0111, 0.6284, 0.3975, 0.4886, 0.2818, 0.1183, 0.3859, 0.0772
0.3675, 0.1739, 0.0982, 0.1716, 0.0678, 0.1227, 0.1809, 0.1643, 0.1517, 0.1001, 0.0121, 0.039
0.0267, 0.024, 0.0218, 0.0094, 0.2329, 0.1158, 0.2193, 0.1005, 0.0621, 0.1082, 0.0428, 0.008
0.1717, 0.0793, 0.08, 0.0698, 0.0756, 0.1048, 0.1212, 0.1286, 0.1006, 0.0142, 0.7287, 0.594
0.3399, 0.2, 0.0134, 0.4656, 0.4473, 0.5293, 0.3293, 0.109, 0.1543, 0.094, 0.1733, 0.2794,
0.4413, 0.7639, 0.2276, 0.6271, 0.7475, 0.6233, 0.5768, 0.3377, 0.0139, 0.1691, 0.0931, 0.1388
0.0372, 0.0101, 0.0662, 0.8172, 0.811, 0.7234, 0.3852, 0.4071, 0.2745, 0.4098, 0.5398, 0.43
0.202, 0.0593, 0.1556, 0.2813, 0.2195, 0.1783, 0.1067, 0.0113, 0.4366, 0.3342, 0.5473, 0.286
7, 0.0121, 0.0296, 0.022, 0.0325, 0.0211, 0.0196, 0.0329, 0.0157, 0.0258, 0.0325, 0.0354, 0.0
12, 0.0475, 0.1054, 0.1662, 0.1234, 0.1087, 0.0541, 0.0111, 0.1173, 0.0567, 0.0876, 0.0469, 0.
0097, 0.7125, 0.4913, 0.5611, 0.3942, 0.1357, 0.2875, 0.0903, 0.2352, 0.2761, 0.2107, 0.238,
0.0434, 0.0913, 0.1626, 0.1435, 0.1168, 0.1117, 0.0115, 0.8774, 0.0609, 0.9207, 0.8318, 0.78
73, 0.7561, 0.7226, 0.7601, 0.6156, 0.4267, 0.42, 0.2762, 0.3865, 0.6462, 0.4395, 0.4787, 0.2
59, 0.0219, 0.0248, 0.0289, 0.0264, 0.0237, 0.0095, 0.425, 0.2081, 0.2961, 0.1564, 0.0693, 0.
717, 0.164, 0.2693, 0.1337, 0.0775, 0.2305, 0.0511, 0.1192, 0.1933, 0.1696, 0.1377, 0.1669, 0.
1412, 0.2467, 0.2051, 0.1745, 0.1334, 0.0122, 0.2146, 0.1515, 0.2848, 0.1254, 0.1026, 0.1539,
0.0244, 0.0394, 0.0236, 0.0191, 0.0226, 0.0161, 0.0224, 0.0269, 0.0293, 0.0252, 0.02, 0.0093,
71, 0.1331, 0.0628, 0.053, 0.0792, 0.0327, 0.0591, 0.1034, 0.0997, 0.0728, 0.0638, 0.0107, 0.
448, 0.0422, 0.0316, 0.0281, 0.0096, 0.1533, 0.1134, 0.2176, 0.0876, 0.0757, 0.1183, 0.0433,
0.3579, 0.1774, 0.0992, 0.121, 0.0636, 0.1117, 0.212, 0.1661, 0.1357, 0.0812, 0.0115, 0.0874,
0.1277, 0.0898, 0.079, 0.0377]
Score: 0.173
Result: DIFFERENT AUTHORS X

--- Summary: 0 same, 1 different ---
(.venv) PS B:\Projects\Current\Python\AuthorIdentification>

```

Рисунок 4.5 – Вивід команди із параметрами --output text --progress

4.2 Захист розробленого програмного забезпечення

Згідно закону України від 23.12.93 р. № 3792-ХІІ “Про авторське право і суміжні права” в редакції від 16.07.01 року визначено терміни “комп’ютерна програма”. Відповідно до законодавства, комп’ютерні програми є об’єктом авторського права, тобто інтелектуальної власності. Основними елементами захисту програмного забезпечення є гарантування конфіденційності, цілісності та доступності даних і функціонування пристрою. [43]

Але особливим нюансом є використання іншого програмного забезпечення (бібліотек) у вихідному коді. Тому слід детально розібратись із кожною із них.

Бібліотеки, що використовуються у проекті:

- PyTorch – головний фреймворк для машинного навчання у проекті, на ньому побудована сіамська нейронна мережа, використовується під час інференсу та навчання. Це open-source бібліотека із BSD-3-Clause ліцензією,

що дозволяє використання у академічних та комерційних проектах;

- NumPy – бібліотека для числових обчислень, що забезпечує роботу із тензорами (частково). Використовується для підготовки даних, обчислення метрик, генерації випадкових чисел. Це open-source бібліотека із BSD-3-Clause ліцензією, що дозволяє використання у академічних та комерційних проектах [44];

- PyYAML – забезпечує парсинг однойменного формату для конфігураційних файлів, файлів фрагментів тексту. Це open-source бібліотека з ліцензією MIT – одна з найбільш пермісивних ліцензій без обмежень на використання [45];

- Scikit-Learn – це бібліотека для машинного навчання, за участі якої проводяться метрики оцінки якості моделі (accuracy, precision, recall, F1, ROC-AUC) та розбиття даних на вибірки. Забезпечує стандартизовану оцінку продуктивності нейронної мережі. Це open-source бібліотека із BSD-3-Clause ліцензією, що дозволяє використання у академічних та комерційних проектах [46];

- Matplotlib – це бібліотека для візуалізації даних та побудови графіків. Використовується для отримання детального графічного представлення процесу навчання. Це open-source бібліотека з ліцензією Python Software Foundation, яка є BSD-сумісною та пермісивною [47];

- BeautifulSoup – це бібліотека для парсингу HTML-файлів. Застосовується для «неочищених» файлів із розміткою, тобто під час препроцесингу. Це open-source бібліотека з ліцензією MIT – одна з найбільш пермісивних ліцензій без обмежень на використання [48];

- requests – це бібліотека для HTTP-запитів. Використовується для завантаження текстів. Це open-source бібліотека з ліцензією Apache 2.0, що дозволяє вільне використання, модифікацію та розповсюдження [49];

- ollama – це бібліотека для взаємодії із LLM через однойменний додаток. Використовується для перейменування книг. Це open-source

бібліотека з ліцензією MIT – одна з найбільш пермісивних ліцензій без обмежень на використання.

Усі вищезазначені ліцензії є open-source та не містять обмежень, що дозволяє вільно використовувати їх у дипломному проєкті без юридичних перешкод.

К6ПЗ_2026

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

77

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Під час впровадження розробленої системи в промислову експлуатацію потрібний чіткий алгоритм дій. Ймовірно, основним клієнтом подібних систем будуть саме правоохоронні структури, отож методика повинна розраховуватись із врахуванням специфіки даних інститутів. Водночас, не потрібно применшувати інші сфери людської діяльності, де може застосовуватись це ПЗ: журналістику, освіти тощо.

Спочатку проведемо аналіз потреб конкретного підрозділу. Правоохоронні структури мають різну специфіку роботи. Один підрозділ може аналізувати повідомлення у соц.мережах, на форумах (для пошуку кіберзлочинців, наприклад), а інший аналізує злочини, що можуть стосуватись серійних вбивць. Для роботи із даним ПЗ потрібні базові навички роботи із терміналом (PowerShell, GNOME-Terminal, Alacritty чи інші). Потрібно визначити, що буде ПЗ працювати на ЕОМ оператора чи на виділеному сервері.

Наступною фазою можна виділити підготовку АЗ, бо система побудована на базі PyTorch та підтримує автоматичне визначення обчислювального пристрою. За наявності відеокарти із підтримкою CUDA обчислення будуть відбуватись значно швидше. Водночас, для використання (а не навчання моделі) достатньо буде звичайного CPU. Для правоохоронних структур це означає, що робоча станція оператора не обов'язково повинна мати потужну відеокарту.

Надалі потрібно визначити програмне забезпечення та дані, що будуть необхідними для коректної роботи системи:

- Файл моделі у форматі PTH;
- Конфігураційний файл у форматі YAML;
- Встановлена платформа CUDA (якщо є потреба у використанні

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

78

графічного процесору);

- Операційна система Windows 11 або ж Ubuntu 24.04 та вище.

Файл моделі є самодостатнім, оскільки містить у собі ваги нейронної мережі, словник токенизації та гіперпараметри. Проте частина даних міститься у YAML-файлі конфігурації. Про це поговоримо у наступному абзаці.

Важливим етапом конфігурації буде налаштування файлу YAML. Для правоохоронних структур може знадобитись змінити поріг класифікації. За замовчуванням він становить 0,65. Окрім цього, може знадобитись змінити метод агрегації результатів: арифметичне середнє, геометричне середнє, гармонічне середнє або медіану [50].

Після налаштування слід провести тестування на контрольних зразках, щоб зрозуміти як модель працює із текстами, які потрібні конкретному підрозділу. Модель навчалась винятково на українськомовних текстах (і трохи суржику), тому аналізувати матеріали на інших мовах не представляється можливим (або з дуже великими похибками).

Слід зауважити, що система має термінальний інтерфейс, що дозволяє приймати файли та повертати результати у декількох форматах:

- TXT – формат для безпосереднього аналізу оператором, що містить вердикт та ймовірність збігу авторства;
- JSON – машинний формат, що зручний для передачі результатів між веб-API для подальшої обробки;
- XML – структурований формат із метаданими, який може використовуватись для формування звітів.

Окремо слід зауважити, що система працює виключно локально. Це є дуже великою перевагою, оскільки жодні дані не передаються на зовнішні сервери. Увесь процес аналізу та винесення вердикту відбувається на звичайному ПК. Матеріали слідства у такому випадку не входять за межі контрольованого середовища на відміну, наприклад, від хмарних рішень або веб-додатків на кшталт TextAttributor. Водночас, цього недостатньо, тому що

Вим	Анк	№ докум.	Підпис	Дата

потрібно забезпечити фізичний та програмний захист ЕОМ, на якій розгорнуто систему, згідно із вимогами нормативних документів. В іншому випадку є загроза витоку даних через засоби соціальної інженерії.

Навчання персоналу є обов’язковою умовою експлуатації. Оператор робочої станції повинен розуміти принципи роботи із командним рядком, вміти формувати вхідні файли у потрібному форматі, а також інтерпретувати результати. Важливо персоналу пояснити, що система видає ймовірність збігу авторства, а не абсолютну відповідь «в останній інстанції». Система ідентифікації авторства тексту – це один із інструментів, що допомагає звузити коло підозрюваних та підтверджувати гіпотези (або їх спростовувати). Порогове значення впливає на баланс між повнотою та точністю. Приклад взаємодії із програмним забезпеченням (отримання довідки про функціонал та термінальні команди) зазначений на рисунку 5.1. При такому підході окремо демонструється копірайт-інформація. Там зазначено розробника, університет, кафедру, наукового керівника тощо.

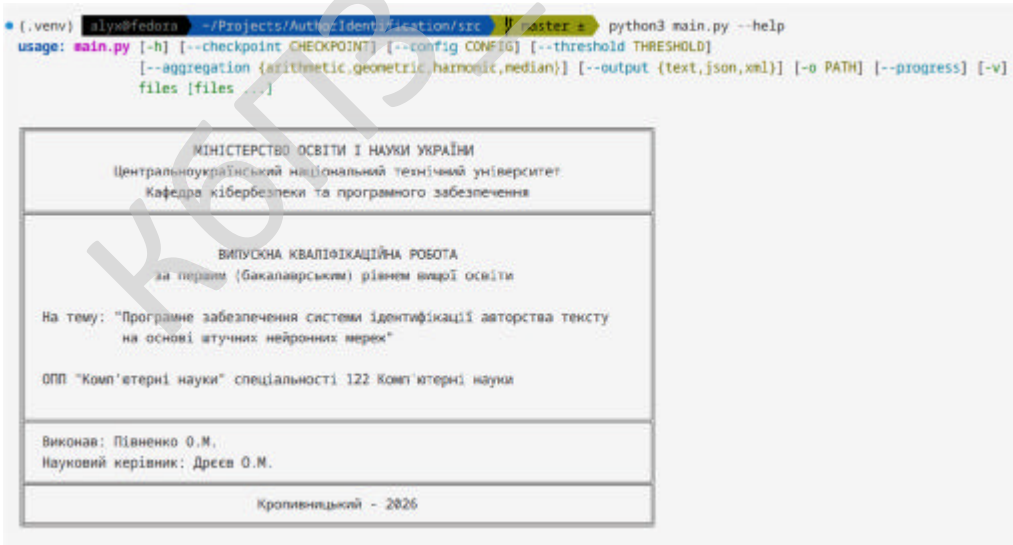
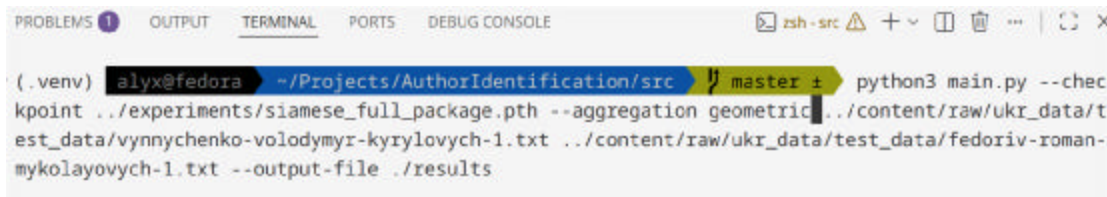


Рисунок 5.1 – Приклад термінальної команди для отримання довідки

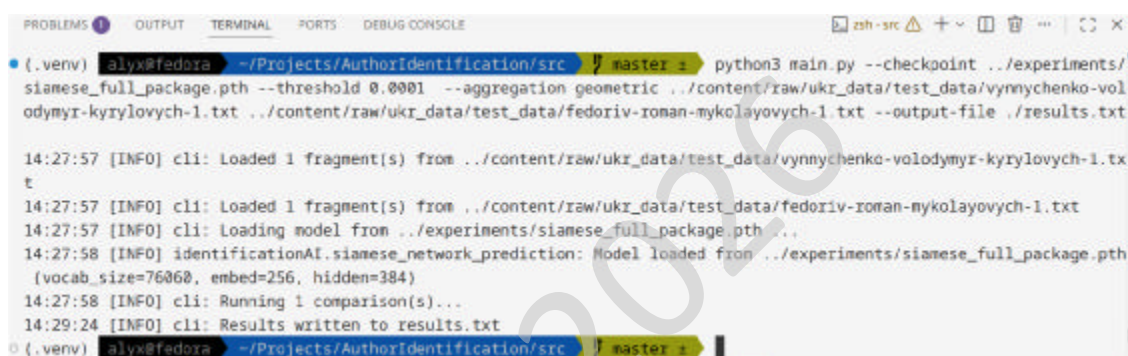
Наступним етапом після отримання довідкової інформації, зазвичай, може бути аналіз фрагментів текстів. Для такого випадку програмне забезпечення представляє параметри, завдяки яким можна провести

конфігурацію. Приклад термінальної команди із режимом агрегації знаходиться на рисунку 5.2, а приклад із пороговим значенням на рисунку 5.3.



```
(.venv) alyx@fedora ~/Projects/AuthorIdentification/src master : python3 main.py --checkpoint ../experiments/siamese_full_package.pth --aggregation geometric ../content/raw/ukr_data/test_data/vynnychenko-volodymyr-kyrylovych-1.txt ../content/raw/ukr_data/test_data/fedoriv-roman-mykolayovych-1.txt --output-file ./results
```

Рисунок 5.2 – Термінальна команда із заданням методу агрегації для великих текстів

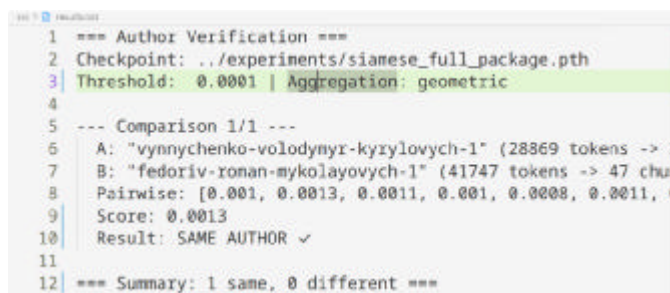


```
(.venv) alyx@fedora ~/Projects/AuthorIdentification/src master : python3 main.py --checkpoint ../experiments/siamese_full_package.pth --threshold 0.0001 --aggregation geometric ../content/raw/ukr_data/test_data/vynnychenko-volodymyr-kyrylovych-1.txt ../content/raw/ukr_data/test_data/fedoriv-roman-mykolayovych-1.txt --output-file ./results.txt
```

```
14:27:57 [INFO] cli: Loaded 1 fragment(s) from ../content/raw/ukr_data/test_data/vynnychenko-volodymyr-kyrylovych-1.txt
14:27:57 [INFO] cli: Loaded 1 fragment(s) from ../content/raw/ukr_data/test_data/fedoriv-roman-mykolayovych-1.txt
14:27:57 [INFO] cli: Loading model from ../experiments/siamese_full_package.pth ...
14:27:58 [INFO] identificationAI.siamese_network_prediction: Model loaded from ../experiments/siamese_full_package.pth
(vocab_size=76060, embed=256, hidden=384)
14:27:58 [INFO] cli: Running 1 comparison(s)...
14:29:24 [INFO] cli: Results written to results.txt
```

Рисунок 5.3 – Термінальна команда із заданням порогового значення (у даному контексті одна десятитисячна)

Третій рисунок також яскраво демонструє вищенаведену проблему, що пов'язана із вибором порогового значення. В даному випадку було обрано занадто мале раціональне число, тому це призвело до того, що майже будь-які автори будуть вважатись за одного. Результат аналізу зазначений на рисунку 5.4. Там зазначена основна інформація, логічне та дійсне значення.



```
1 === Author Verification ===
2 Checkpoint: ../experiments/siamese_full_package.pth
3 Threshold: 0.0001 | Aggregation: geometric
4
5 --- Comparison 1/1 ---
6 A: "vynnychenko-volodymyr-kyrylovych-1" (28869 tokens -> 3
7 B: "fedoriv-roman-mykolayovych-1" (41747 tokens -> 47 chunk
8 Pairwise: [0.001, 0.0013, 0.0011, 0.001, 0.0008, 0.0011, 0
9 Score: 0.0013
10 Result: SAME AUTHOR ✓
11
12 === Summary: 1 same, 0 different ===
```

Рисунок 5.4 – Результат аналізу в текстовому файлі

Інформація про розробника, мету створення, тему, університет, кафедру та юридичні обмеження зазначено на рисунку 5.5. Ця інформація винесена окремою функцією, що повертає значення типу «рядок».

```

24 def copyright_notify_string():
25     return """
26
27         МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
28         Центральноукраїнський національний технічний університет
29         Кафедра кібербезпеки та програмного забезпечення
30
31
32         ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
33         за першим (бакалаврським) рівнем вищої освіти
34
35         На тему: "Програмне забезпечення системи ідентифікації авторства тексту
36                 на основі штучних нейронних мереж"
37
38         ОПП "Комп'ютерні науки" спеціальності 122 Комп'ютерні науки
39
40
41         Виконав: Півненко О.М.
42         Науковий керівник: Дреєв О.М.
43
44         Кропивницький - 2026
45
46     """

```

Рисунок 5.5 – Авторське право на розробку

Серед функціональних можливостей присутня також демонстрація прогресу аналізу фрагментів. Продемонстровано на рисунку 5.6. На ньому зображено фрагмент, де проведено аналіз чотирнадцяти відсотків пар.

```

python3 main.py --checkpoint ../experiments/simase_full_package.pth --progress --threshold 0.70 --aggre
tion geometric ../content/raw/ukr_data/test_data/vynnychenko-volodymyr-kyrylovych-1.txt ../content/raw/ukr_data/test_data/fedoriv-roman-mykolayovych-1.txt --output-file ../results.tx
t
15:05:02 [INFO] cli: Loaded 1 fragment[s] from ../content/raw/ukr_data/test_data/vynnychenko-volodymyr-kyrylovych-1.txt
15:05:02 [INFO] cli: Loaded 1 fragment[s] from ../content/raw/ukr_data/test_data/fedoriv-roman-mykolayovych-1.txt
15:05:02 [INFO] cli: Loading model from ../experiments/simase_full_package.pth ...
15:05:02 [INFO] identification/simase_network_prediction: Model loaded from ../experiments/simase_full_package.pth (vocab_size=76868, embed=256, hidden=384)
15:05:02 [INFO] cli: Running 1 comparison(s)...
[1/1] Comparing "vynnychenko-volodymyr-kyrylovych-1" vs "fedoriv-roman-mykolayovych-1"...
Chunks: A=33, B=47 -> 1551 pairwise comparison[s]
Pair [ 215/1551] 13.9%

```

Рисунок 5.6 – Аналіз фрагментів із демонстрацією прогресу

6 ОСНОВНІ ВИСНОВКИ

Отже, у ході виконання бакалаврської випускної кваліфікаційної роботи була створена система ідентифікації авторства тексту. Вона була реалізована за допомогою мови програмування Python, фреймворку PyTorch, платформи CUDA, а також інших модулів та бібліотек. Основний спосіб ідентифікації – це використання нейромережевого підходу. Він відрізняється від статистичних підходів тим, що здатний обробляти природню мову, зберігати інформацію про контекст, сприймати стилістичні особливості тексту, проводити аналіз вхідних фрагментів тексту двонаправлено, при цьому зберігаючи «розумні» вимоги до обчислювальних ресурсів. Хоча, безумовно, статистичні підходи є більш обчислювально ефективними у порівнянні із тим, що був реалізований мною.

Були проаналізовані конкуренти, аналогічні системи (або дуже схожі), що дали уявлення про те, як робити власне ПЗ. Під час проектування, а також реалізації, було враховано усі переваги та недоліки; зроблено припущення, що дозволили максимально ефективно організувати власне програмне забезпечення.

Система містить у собі CLI-інтерфейс для взаємодії із нейромережею, дозволяє обробляти файли у форматах YAML, JSON, TXT, зберігати інформацію про інференс у тих же самих форматах. Додаток може видавати інформацію про свої параметри, можливості, а також приклад виклику. Це облегшує взаємодію із утилітою.

Створено вихідний код, що дозволяє навчати нейронну мережу. Під час навчання нейромережі можна динамічно налаштувати параметри, не змінюючи при цьому вихідний код (за допомогою файлу в форматі YAML), що пов'язаний безпосередньо із тренуванням. Присутній у системі модуль, що дозволяє проводити парсинг інформації із веб-джерел (Ukrlib та Project

Gutenberg), структурувати її та очищувати для подальшого використання. Його особливістю є те, що він використовує для організації іменування файлів локальну нейронну мережу (або хмарну, якщо це дозволяє версія платформи Ollama). Це дозволило вирішити проблему складноструктурованих даних (імен файлів), оскільки вони не піддавались аналізу за допомогою регулярних виразів та класичних алгоритмів (імена файлів могли містити як латинські, так і кириличні літери, мали різну довжину, різні символи-роздільники, різні числові індекси тощо). Важливу роль в обробці таких даних зіграв новий сервіс Ollama Cloud, що дозволив використовувати нейронні мережі, що розгортаються у хмарній інфраструктурі, але конкретно у цьому випадку йде мова не пропріетарні моделі на кшталт GPT-4o, Claude Sonnet 4, а про open-source моделі, але які важко запустити на локальному персональному комп'ютері через брак обчислювальних ресурсів.

Загалом, створена система виявилось доволі точною (приблизно 92% на відомих авторах, але невідомих текстах, та ~89% для невідомих авторів на невідомих текстах). Вона працює із фрагментами середнього розміру (1000 символів). Це дозволяє її використовувати як для аналізу великих публікацій, так і для малих повідомлень. Додаток дозволяє масштабувати аналіз навіть у тому випадку, якщо вхідні тексти більші або менші від встановленої кількості символів. Він має зручний термінальний інтерфейс, який хоча і розходиться із мейнстрімним дизайном сучасних додатків, але дає швидкий, ефективний інтерфейс комунікації користувачу.

Під час проектування було використано багато відкритих джерел: брались до уваги роботи відомих вчених (Ле Куна, що зараз є головним науковцем у питаннях ШІ в компанії Meta, і завдяки його роботі було вирішено використати сіамську нейронну мережу; Шмідхубера, що опублікував роботи по темі рекурентних нейронних мереж із довгостроковою пам'яттю, тобто LSTM, які були використані навіть у продуктах компанії Google; Франка

Хуттера, що під час виступів на конференціях популяризував ідею вагової регуляризації), офіційна документація Python, PyTorch, Ollama, Ubuntu.

Створений додаток відповідає поставленим вимогам, використовує сучасні архітектурні, програмні та математичні підходи.

Водночас, система має подвійне призначення: вона може використовуватись як для «благородних цілей», так і для пригнічення прав та свобод. Цей аспект використання залежить винятково від суб'єктів, що будуть користуватись цим ПЗ.

КБПЗ_2026

Вим	Анк	№ докум.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

85

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. UnaBomber // FBI Agency [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://www.fbi.gov/history/famous-cases/unabomber>
2. Saving and Loading Models [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: https://docs.pytorch.org/tutorials/beginner/saving_loading_models.html
3. Github – ROCm: AMD ROCm Software [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://github.com/ROCm/ROCm>
4. Countries and Territories | Freedom House [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://freedomhouse.org/country/scores>
5. Eastern District of Virginia | United States File Civil Lawsuit Against Edward Snowden // U.S. Department of JUSTICE [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://www.justice.gov/usao-edva/pr/united-states-files-civil-lawsuit-against-edward-snowden>
6. Edward Snowden: Leaks that exposed US spy programme [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://www.bbc.com/news/world-us-canada-23123964>
7. Softpedia about JGAAP [Електронний ресурс]. – Режим доступу: https://www.softpedia.com/get/Science-CAD/JGAAP.shtml#sgal_0
8. The State of Coding the Future of Java And AI [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://devblogs.microsoft.com/java/the-state-of-coding-the-future-with-java-and-ai/>
9. NeoNeuro Company Contact Information [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://neoneuro.com/contactus>
10. NeoNeuro AuthorShip Attribution Software Product [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://neoneuro.com/products/authorship-attribution>
11. Desktop Windows Version Market Share Worldwide [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу:

<https://gs.statcounter.com/windows-version-market-share/desktop/worldwide/#monthly-201801-201812-bar>

12. Text Attributor [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <http://ta.mova.info/>

13. Text Attributor Instruction [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <http://ta.mova.info/instructions>

14. Vim Emulation in VS Code [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://marketplace.visualstudio.com/items?itemName=vscodvim.vim>

15. NvChad Framework [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://nvchad.com/>

16. Marketplace Visual Studio [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://marketplace.visualstudio.com/items?itemName=ms-pyright.pyright>

17. Jupyter Notebook [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter>

18. Benchmarking TPU, GPU and CPU Platforms for Deep Learning [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://arxiv.org/pdf/1907.10701>

19. Google Cloud Tensor Processing Unit [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://cloud.google.com/tpu>

20. Радіо Свобода [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://www.svoboda.org/a/tramp-vnovj-prigrozil-sanktsiyami-esli-ne-budet-mirav-ukraine/33513725.html>

21. CUDA Platform for Accelerated Computing [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://developer.nvidia.com/cuda>

22. AMD ROCm Documentation [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://rocm.docs.amd.com/en/latest/>

23. Intel OneAPI Documentation [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://www.intel.com/content/www/us/en/developer/tools/oneapi/documentation.html>

Вим	Анк	№ докум.	Підпис	Дата

24. Tensorflow Ecosystem [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://www.tensorflow.org/resources/learn-ml>
25. Keras API Documentation [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://keras.io/api/>
26. Project PyTorch [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://pytorch.org/projects/pytorch/>
27. JAX [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://docs.jax.dev/en/latest/faq.html>
28. MXNet [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://mxnet.apache.org/versions/1.9.1/>
29. Ollama documentation [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://docs.ollama.com/>
30. Ollama Python Library [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://github.com/ollama/ollama-python>
31. Ollama Cloud [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://ollama.com/pricing>
32. Pathlib – Object-oriented filesystem paths [ЕЛЕКТРОННИЙ РЕСУРС]. – режим доступу: <https://docs.python.org/uk/3/library/pathlib.html>
33. Weerasinghe J., Greenstadt R. Siamese Networks for Large-Scale Author Identification. 2019. arXiv:1912.10616.
34. Graves A., Schmidhuber J. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. // Neural Networks. 2005. Vol. 18, No. 5–6. P. 602–610. DOI: 10.1016/j.neunet.2005.06.042 [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: https://www.cs.toronto.edu/~graves/nn_2005.pdf
35. Bromley J., Guyon I., LeCun Y., Säckinger E., Shah R. Signature Verification using a "Siamese" Time Delay Neural Network. // Advances in Neural Information Processing Systems (NeurIPS). 1993. Vol. 6. P. 737–744.
36. Mou L., Men R., Li G., Xu Y., Zhang L., Yan R., Jin Z. Natural

Language Inference by Tree-Based Convolution and Heuristic Matching. // Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL). 2016. P. 130–136. DOI: 10.18653/v1/P16-2022.

37. Chen Q., Zhu X., Ling Z., Wei S., Jiang H., Inkpen D. Enhanced LSTM for Natural Language Inference. // Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL). 2017. P. 1657–1668. DOI: 10.18653/v1/P17-1152.

38. Nair V., Hinton G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. // Proceedings of the 27th International Conference on Machine Learning (ICML). 2010. P. 807–814.

39. Loshchilov I., Hutter F. Decoupled Weight Decay Regularization. // Proceedings of the 7th International Conference on Learning Representations (ICLR). 2019. arXiv:1711.05101.

40. Pascanu R., Mikolov T., Bengio Y. On the Difficulty of Training Recurrent Neural Networks. // Proceedings of the 30th International Conference on Machine Learning (ICML). 2013. P. 1310–1318.

41. Fawcett T. An Introduction to ROC Analysis. Pattern Recognition Letters. 2006. Vol. 27, No. 8. P. 861–874. DOI: 10.1016/j.patrec.2005.10.010.

42. Modular Monoliths Are a Good Idea // HackerNews [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://news.ycombinator.com/item?id=41534179>

43. Про авторське право і суміжні права [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/2811-20#Text>

44. NumPy License [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://numpy.org/doc/stable/license.html>

45. PyYAML [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://github.com/yaml/pyyaml>

46. Scikit-Learn License [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу: <https://github.com/scikit-learn/scikit-learn/blob/main/COPYING>

47. Matplotlib License [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу:

<https://matplotlib.org/stable/project/license.html>

48. BeautifulSoup License [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу:
<https://github.com/akalongman/python-beautifulsoup/blob/master/LICENSE>

49. Requests License [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу:
<https://github.com/psf/requests?tab=Apache-2.0-1-ov-file#readme>

50. Ziou D. Pythagorean Centrality for Data Selection. arXiv. 2023. arXiv:2301.10010. [ЕЛЕКТРОННИЙ РЕСУРС]. – Режим доступу:
<https://arxiv.org/abs/2301.10010>

К6ПЗ_2026

Вим	Анк	№ доквм.	Підпис	Дата

ВКРБ-122.26.0025.00.00.ПЗ

Арк.

90