НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2021 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «*Сервіс з розпізнавання іменованих сутностей в текстах*»

Виконав: студент *IV* курсу, групи *ІС-72*

Кривохижа Роман Андрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *ст.в., к.т.н. Олійник Юрій Олександрович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Новінський Валерій Петрович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доц. кафедри ТК, к.т.н., доц. Ткач М.М.* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2021 р.

З А В Д А Н Н Я
на дипломний проєкт студенту

Кривохижі Роману Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Сервіс з розпізнавання іменованих сутностей в текстах*»,

керівник проєкту Олійник Юрій Олександрович, *к.т.н.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “ *11* ” *травня* 2021 р. № 1139-с

2. Термін подання студентом проєкту “*04*”*червня* 2021 року

3. Вихідні дані до проєкту

Технічне завдання

Побудувати сервіс з іменований сутностей для користувачів. Створити модель, використовуючи інформативні векторні представлення текстів за допомогою моделі BERT. Розробити веб-застосунок для візуалізації результатів роботи моделі.

4. Зміст пояснювальної записки

1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі

2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних

3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв'язання

4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна діяльності

2. Схема структурна компонентів програмного забезпечення

3. Схема структурна залежності модулів

4. Схема структурна класів програмного забезпечення

5. Схема структурна розгортання

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «*7*» *квітня 2021 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*17.04.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*23.04.2021*	
3.	*Постановка та формалізація задачі*	*23.04.2021*	
4.	*Розробка інформаційного забезпечення*	*30.04.2021*	
5.	*Алгоритмізація задачі*	*04.05.2021*	
6.	*Обґрунтування використовуваних технічних засобів*	*10.05.2021*	
7.	*Розробка програмного забезпечення*	*14.05.2021*	
8.	*Налагодження програми*	*01.06.2021*	
9.	*Виконання графічних документів*	*01.06.2021*	
10.	*Оформлення пояснювальної записки*	*15.05.2021*	
11.	*Подання ДП на попередній захист*	*14.05.2021*	
12.	*Подання ДП на основний захист*	*04.06.2021*	
13.	*Подання ДП рецензенту*	*07.06.2021*	

Студент Роман КРИВОХИЖА

Керівник Юрій ОЛІЙНИК

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 7217.00.000 ПЗ	Пояснювальна записка	92	
3	А4	ДП 7217.01.000 ТЗ	Технічне завдання	11	
4	А3	ДП 7217.02.000 ССД	Схема структурна діяльності	1	
5	А3	ДП 7217.03.000 ССК	Схема структурна класів програмного	1	
			забезпечення		
6	А3	ДП 7217.04.000 ССЗ	Схема структурна залежності модулів	1	
7	А3	ДП 7217.06.000 ССК	Схема структурна компонентів	1	
8	А3	ДП 7217.07.000 ССР	Схема структурна розгортання	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 7217.00.000.00		
Розробн.		Кривохижа Р. А.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Олійник Ю. О.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-72	
Н/контр.		Новінський В.П.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: «Сервіс з розпізнавання іменованих сутностей в текстах»

Київ – 2021 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 29 рисунків, 7 таблиць, 1 додаток, 26 джерел.

Дипломний проєкт присвячений розробці сервісу з розпізнавання іменованих сутностей.

У розділі інформаційного забезпечення було описано вхідні та вихідні дані даного проєкту, їх природу, як саме була сформована навчальна та валідаційна вибірка, описано процес підготовки даних для можливості їх передачі в модель нейронної мережі.

Розділ математичного забезпечення присвячений опису можливих підходів до вирішення поставленої задачі. Також, в цьому розділі обґрунтовується вибір обраного підходу.

У розділі з програмного забезпечення описуються технічні вимоги до системи, засоби розробки та архітектуру програмного забезпечення.

У технологічному розділі описана інструкція користувача та проведено тестування розробленого сервісу.

НЕЙРОННІ МЕРЕЖІ, МАШИННЕ НАВЧАННЯ, МОВНІ МОДЕЛІ, КЛАСИФІКАЦІЯ ТОКЕНІВ, ТРАНСФОРМЕРИ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of 5 sections, contains 29 pictures, 7 tables, 1 appendix, 26 sources.

The diploma project is concentrated on the creation of the service for the recognition of named entities.

In the section of information, the input and the output data of the project and their nature are described. Described the training, validation sample and the process of data preparation for the possibility of their transfer to the network model.

The section of mathematical software is concentrated on description of possible approaches to the solution of the task. Furthermore, in this section justifies the choice of the chosen approach.

The software section describes the technical requirements of the system, software development tools and architecture.

The technological section presents the user manual and testing of the developed service.

NEURAL NETWORKS, MACHINE LEARNING, LANGUAGE MODELS, TOKEN CLASSIFICATION, TRANSFORMERS.

ЗМІСТ

ВСТУП **5**

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ **7**

1.1 Опис предметного середовища 7

1.1.1 Опис процесу діяльності *7*

1.1.2 Опис функціональної моделі *8*

1.2 Огляд наявних аналогів 9

1.3 Постановка задачі 12

1.3.1 Призначення розробки *12*

1.3.2 Цілі та задачі розробки *13*

Висновок до розділу 14

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ **16**

2.1 Вхідні дані 16

2.2 Вихідні дані 18

2.3 Опис структури бази даних 19

2.4 Структура масивів інформації 20

Висновок до розділу 20

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ **21**

3.1 Змістовна постановка задачі 21

3.2 Математична постановка задачі 22

3.1 Обґрунтування методу розв'язання 23

3.1.1 Обґрунтування вибору нейронної мережі *23*

3.1.2 Обґрунтування вибору функції витрат *29*

3.1.3 Обґрунтування вибору базової архітектури нейронної мережі *30*

3.1.4 Обґрунтування вибору архітектури останнього шару моделі *32*

3.2 Опис методів розв'язання 34

3.2.1 Особливості підготовки даних для навчання *34*

3.2.2 Обрана архітектура нейронної мережі *36*

3.2.3 Особливості етапу навчання нейронної мережі *40*

3.2.4 Результати навчання нейронної мережі *41*

Висновок до розділу 42

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ **43**

4.1 Засоби розробки 43
4.2 Вимоги до технічного забезпечення 45
4.3 Архітектура програмного забезпечення 46
4.3.1 Діаграма компонентів *46*
4.3.2 Діаграма класів *46*
4.3.3 Діаграма залежності пакетів *47*
4.3.4 Діаграма розгортання *47*
4.3.5 Специфікація функцій *47*
Висновок до розділу 49
5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ **50**
5.1 Керівництво користувача 50
5.2 Випробування програмного продукту 52
5.2.1 Мета випробувань *52*
5.2.2 Загальні положення *52*
5.2.3 Результати випробувань *52*
Висновок до розділу 54
ЗАГАЛЬНІ ВИСНОВКИ **55**
ПЕРЕЛІК ПОСИЛАНЬ **58**
ДОДАТОК А **61**

ВСТУП

Дипломний проєкт присвячений розробці сервісу з розпізнавання іменованих сутностей в текстах для української мови, використовуючи останні SOTA (state-of-the-art) моделі BERT [1] (Bidirectional Encoder Representations from Transformers) для роботи з текстовими даними.

Щодня, людство генерує величезну кількість даних, які можуть бути як структурованими, так і неструктурованими. Зі структурованими даними досить легко працювати, до них можна застосувати певні аналітичні інструменти чи використовувати їх в якості бази знань. А ось із неструктурованими даними працювати набагато складніше, тому зазвичай використовують допоміжні інструменти, які дають можливість або визначити певну структуру, патерни чи ключові особливості, оскільки такі дані по своїй природі є хаотичними. Також, такий тип даних неможливо зберігати в реляційному вигляді, що робить їх аналіз доволі складним.

Одним з найпоширеніших неструктурованих видів даних є текстові дані. Вони можуть походити з багатьох джерел, таких як: декларації, чеки, електронні листи, файли з логами певних сервісів, частини книг, статей, сайтів, діалогів, тощо. Одним з аналітичних інструментів для аналізу текстових даних є алгоритм розпізнавання іменованих сутностей [2]. В залежності від доменної області чи сфери застосування, набір сутностей, які ми намагаємось виявляти, може відрізнятись. В класичній постановці задачі, такими сутностями є: локації, персони та організації.

В останні роки сфера машинного навчання та глибокого машинного навчання далеко просунулась в сфері побудови інформативних векторних представлень слів та текстів. Одним останніх підходів, які вважаються SOTA (state-of-the-art) є моделі BERT [1] (Bidirectional Encoder Representations from Transformers). В ході навчання, ці моделі вирішують одночасно декілька задач: маскування слів та класифікації. Це змушує моделі розуміти контексти, в яких

зустрічаються слова, що в свою чергу породжує їх інформативне векторне представлення.

Практичне значення одержаних результатів. Розроблено алгоритм з розпізнавання іменованих сутностей в текстах з підтримкою української мови.

Публікації. Результати роботи були опубліковані у тезах доповідей на науково-технічних конференціях кафедри АСОІУ.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

За деякими оцінками, станом на 1998 рік, кількість неструктурованих даних досягала близько 80-90% від загальної кількості [3]. На сьогоднішній день, з розвитком інфраструктури та наявності можливостей для зберігання величезних об’ємів даних, зріс як і даний показних, так і загальний об’єм неструктурованих даних.

Доволі поширеними є неструктуровані текстові дані. Вони можуть походити з багатьох джерел, таких як: декларації, чеки, електронні листи, файли з логами певних сервісів, частини книг, статей, сайтів, діалогів, тощо.

Користі від сирих даних доволі мало, тому використовують різноманітні алгоритми та методи, що допомагають опрацьовувати величезні масиви записів та діставати з них певні знання, що можуть або допомогти вирішити прикладну задачу, або структурувати ці дані.

В контексті роботи з текстами, виділяють алгоритм NER [2] (named entity recognition) – розпізнавання іменованих сутностей. Даний алгоритм дає можливість знайти в тексті певний набір сутностей та виділити їх. Такими сутностями можуть бути: локації, персони, назви організацій, тощо. Надалі, виділивши дані сутності, дані можуть бути або передані для валідації людині, що в подальшому буде приймати рішення, або передані на вхід іншому алгоритму, що продовжить виконувати інтелектуальний аналіз.

Сутностями в тексті може виступати будь-що: назви автомобілів, номера телефонів, назви міст. Все це залежить від доменної області, де застосовується алгоритм та від набору даних.

1.1.1 Опис процесу діяльності

Процес діяльності користувача, який використовує візуальний інтерфейс сервісу з розпізнавання іменованих сутностей, полягає у введені

певного тексту українською мовою, над яким будуть виконані певні перетворення та розбиття на токени, для їх подальшої класифікації.

Після введення користувачем певного тексту, відбувається Word Piece [9] токенизація слів – розбиття слів на підслова. Надалі, отримані токени переводяться в простір індексів, оскільки наші моделі вміють оперувати лише числами, та виконується подальша класифікація токенів.

Загальна діаграма діяльності зображена на рисунку 1.1:

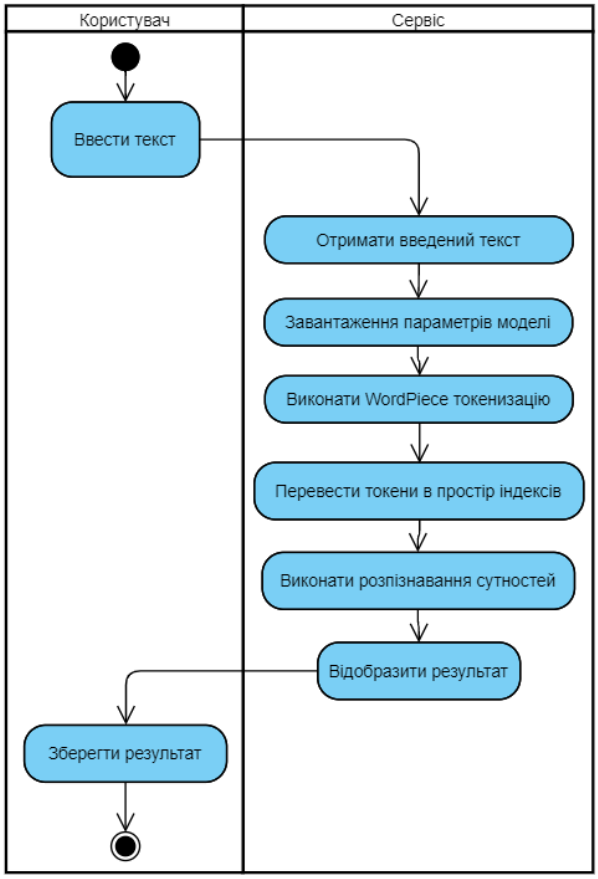


Рисунок 1.1 – Діаграма діяльності системи

1.1.2 Опис функціональної моделі

Основним актором системи є користувач. Йому доступна можливість введення тексту, для якого буде виконана класифікація токенів.

Функціональні вимоги та встановлена їх пріоритетність наведені в таблиці 1.1

Таблиця 1.1 – функціональні вимоги

Актор	Варіант використання	Функціональна вимога	Пріоритет
Користувач	Введення тексту	1. Система надає можливість користувачу заповнити поле для введення тексту	Високий
	Вибір заготовлених прикладів	1. Система надає можливість обрати заготовлені варіанти текстів для подальшого огляду та тестування системи	Високий
Сервіс	Модель розпізнавання іменованих сутностей	1. Після введення тексту система виконує класифікацію токенів	Високий
	Виведення інформації про застосовану модель	1. Система надає можливість перегляну криві процесу навчання та гіперпараметри моделі	Середній
	Збереження історії користування сервісом	1. Система зберігає всю історію запусків моделі та результат роботи до БД	Середній
	Налаштування палітри кольорів для візуального відображення	1. Сервіс дозволяє підібрати зручну кольорову палітру для візуалізації результату	Низький

1.2 Огляд наявних аналогів

Розглянемо найпопулярніші додатки, модулі та бібліотеки, які дозволяють виконувати розпізнавання іменованих сутностей в текстах. В ході аналізу, було виділено наступні аналоги:

- Spacy;
- NLTK;
- Natasha;
- DeepPavlov;
- Ner-ua.

Розглянемо окремо кожен з аналогів:

- Spacy [4] – одна з найвідоміших бібліотек для мови Python [11], яка написана на CPython [10] до робить її досить швидкою. Дана бібліотека є багатофункціональною, має інструменти для вирішення багатьох задач з напрямку NLP. Окрім великої кількості функцій, вона містить навчені моделі та зручне API для їх використання. Основними структурами даних є Vocab та Doc. Vocab дозволяє зберігати спільну інформацію в певних таблицях. Завдяки такому центральному зберіганню інформації, відсутня необхідність робити копії цих даних. Цей функціонал дозволяє суттєво економити пам’ять. В свою чергу, об’єкт Doc допомагає швидко та без зайвих процесів копіювання працювати бібліотеці Spacy з цими даними.
- NLTK [5] – модуль, що написаний на мові програмування Python [11], містить набір бібліотек та програм для роботи із природними мовами. Модуль NLTK містить велику кількість допоміжного функціоналу, такого як:
  а) Tokenizers – сутності, що дозволяють токенизувати текстові дані.
  б) Набори стоп-слів – слова, які доволі часто зустрічаються в природній мови та не несуть ніякої цінності для розуміння контексту.

в) Stemmers – сутності, що дозволяють дають можливість виконувати процес скорочення слів, шляхом відкидання закінчення чи суфікса.

г) Lemmatizers – сутності, що дозволяють проводити нормалізацію слів, шляхом приведення їй до словникової форми.

- Natasha [6] – бібліотека, що підтримує мову Python [11], яка дозволяє вирішувати базові задачі для обробки російської мови. Однією з такої задача є розпізнавання іменованих сутностей. Варто зауважити, що дана бібліотека використовує для своєї роботи останні SOTA результати в сфері NLP, а саме - моделі BERT [1].
- DeepPavlov [7] – це бібліотека з відкритим кодом, що містить велику кількість наборів натренованих нейронних мереж для аналізу тексту та набір різноманітних пайплайнів. Бібліотека націлене на роботу з діалоговими системами, але має і ряд додаткового функціоналу.
- Ner-ua [8] – це модуль з відкритим кодом, що написаний мовою програмування Python [11]. Він вирішує задачу розпізнавання іменованих сутностей в текстах та використовує мовну модель BERT [1].

Як можемо бачити, далеко не всі з наведених вище додатків використовуються останні SOTA підходи до вирішення задач зі сфери NLP. Також, в багатьох відсутня підтримка української мови, що є ключовим пунктом. Дійсно, деякі з додатків показують дуже непогані результати, але вони підтримують використання лише російської мови. В деяких часткових випадках такі моделі можуть бути використані для вирішення певних задач, але не можна бути впевненим в коректності їх роботи. Також, викає проблема зі словником, який використовується для Word Piece [9] токенизації, оскільки

в ньому можуть бути відсутні певні символи та літери, що унеможливить коректне виконання процесу розбиття слів на підслова. Наведемо порівняльну таблицю описаних вище аналогів:

Таблиця 1.2 – порівняння аналогів

Назва аналогу	Підтримка мови Python	Наявність візуального інтерфейсу	Наявність моделей BERT	Підтримка української мови	Постійна підтримка продукту
Spacy	Так	Ні	Ні	Так, але частково	Так
NLTK	Так	Ні	Ні	Так, але частково	Так
Natasha	Так	Ні	Так	Ні	Так
Deep Pavlov	Так	Ні	Так	Ні	Так
Ner-ua	Так	Ні	Так	Так	Ні

1.3 Постановка задачі

1.3.1 Призначення розробки

Даний програмний продукт призначений для розпізнавання іменованих сутностей в текстах, написаних українською мовою. Цей сервіс є допоміжним інструментом при роботі з великими корпусами текстів та різноманітними неструктурованими текстовими даними.

Використання даного методу може полегшити ручний аналіз текстових даних чи бути використаним в якості мета алгоритму для більш складної моделі. Прикладом більш складної моделі може слугувати діалогова система чи система пошуку відповідей в тексті (Question-Answering System).

Розроблений програмний продукт базується на останніх SOTA підходах до роботи з текстовими даними зі сфери NLP – це мовні моделі типу BERT [1]. По суті, дана модель є великою нейронною мережею з великою кількістю параметрів, що вже була навчена на великих корпусах текстів певної мови. Використання даних моделей дає змогу отримати інформативні векторні

представлення слів у тексті, що враховують контекст, в якому зустрічається задане слово. Даний підхід дасть змогу краще зрозуміти алгоритму контекст речення та правильно класифікувати всі токени заданого тексту.

У користувача є можливість ввести певний текст чи обрати один з заготовлених прикладів, який надалі буде переданий алгоритму, що в свою чергу виконає класифікацію токенів, відповідно до поставленої задачі розпізнавання іменованих сутностей. Також, користувач має змогу переглянути історію своїх запитів, результат їх виконання, інформацію про використовувану модель, її параметри та інформацію, зібрану під час процесу навчання.

1.3.2 Цілі та задачі розробки

Мета: покращення розпізнавання іменованих сутностей в текстах для української мови.

Призначення: розробка сервісу для розпізнавання іменованих сутностей в текстах для української мови, з використанням SOTA моделей BERT [1].

В результаті реалізації сервісу очікується покращити результат вирішення задачі розпізнавання іменованих сутностей в текстах та зробити його більш універсальним, за рахунок використання розумного підходу до токенізації слів, а саме – розділення на підслова, використовуючи Work Piece [9] токенізацію.

Для реалізації поставленої мети потрібно реалізувати наступні завдання:

- зібрати дані для формування навчальної та тестової вибірки;
- виконати обробку зібраних даних та привести їх до зручного для подальшого використання вигляду;
- розробити модуль для завантаження та оперування даними;
- розробити модуль для можливості запуску моделей, що вирішують задачу розпізнавання іменованих сутностей в текстах;

- розробити архітектуру моделі для вирішення задачі розпізнавання іменованих сутностей;
- реалізувати підхід fine-tuning [12] моделі типу BERT [1] з навчанням лише останнього шару моделі;
- розробити гнучкий інтерфейс для можливості тестування різних варіантів архітектури останнього шару моделі;
- навчання побудованої нейронної мережі;
- реалізувати збір метрик, значень функції втрат для подальшого аналізу та відображення;
- підбір гіперпараметрів для покращення точності класифікації токенів;
- розробити інтерфейс для взаємодії користувача з розробленою моделлю, в якому він зможе ввести потрібний для класифікації текст чи обрати один з прикладів;
- розробити інтерфейс для відображення результатів навчання моделі;
- розробити інтерфейс для відображення історії користування моделлю;
- дослідити можливість використання алгоритмів дистиляції для зменшення розмірів нейронної мережі з мінімальними втратами точності прогнозування на прикладі задачі з розпізнавання іменованих сутностей в текстах;

Висновок до розділу

В даному розділі було описано предметне середовище інформаційної системи та деталізовано процес діяльності. Також, було визначено основних акторів, їх ролі, модель поведінки та сформовано функціональні вимоги. Окремо, виконано аналіз аналогів та конкурентних програм, що мають подібний функціонал. Визначено їх переваги, недоліки, побудовано

порівняльну таблицю. Також, визначено їх відмінності від розроблюваного сервісу. Поставлено мету та чіткі цілі розробки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідними даними для даного проєкту є:

- вхідні тексти, що знаходяться в наступній директорії «data/corpus» відносно кореня проєкту та мають розширення «tok.txt». Дані тексти зберігаються в токенізованому по словам вигляді, а окремі речення розділені переходом на новий рядок. Переважна більшість цих текстів належить до різноманітних інформаційних видань, статей та художніх творів. Всього, в даному наборі текстів знаходиться 12014 речень. Приклад одного з файлів зображено на рисунку 2.1;

```
Зрозуміло , що український бізнес почав використовувати КСВ як інструмент формування своєї репутації буквально декілька років тому .
З одного боку , саме через це більшість проектів КСВ здійснюються епізодично та деколи виглядають , радше , як просто благодійність .
Винятком будуть хіба що представництва іноземних корпорацій .
З іншого боку , для українських компаній відкривається потужне « вікно можливостей » , щоб втілювати власні унікальні стратегії .

Філантропія як пріоритет

Найпоширенішим підходом українського бізнесу у КСВ є філантропія – компанії виділяють кошти на суспільно важливі проекти .
Це може бути як власна ініціатива , так і відповідь на запит від громади .
Це абсолютно нормальна практика через недостатній рівень статків та доходів переважної більшості українських громадян та громад .
І хоча деякі місцеві експерти вважають це базовим рівнем , що свідчить про недостатній розвиток КСВ , ця діяльність заслуговує на повагу .
Особливо якщо зауважити , що такі проекти часто спрямовані на обладнання шкіл , сприяння освіті , допомогу вимушеним переселенцям або літнім людям , як , наприклад , проект «
Вони абсолютно відповідають Глобальному договору та Цілям сталого розвитку ООН .
```

Рисунок 2.1 – приклад вмісту файлу з текстами

- вхідні сутності, що знаходяться в наступні директорії «data/corpus» відносно кореня проєкту та мають розширення «tok.ann». Файл з анотаціями має табличний вигляд, а розділовим символом є табуляція. В ньому знаходяться наступні колонки:
  а) анотація – сутність, до якої належить слово (в даному наборі даних помічено 4 типи сутностей: локації, персони, організації, інше);
  б) індекс першого символу даної сутності в тексті;
  в) індекс останнього символу даної сутності в тексті;
  г) текстове значення сутності.

Приклад одного з файлів зображено на рисунку 2.2;

```
T1	ORG 1137 1145	Життєлюб
T2	PERS 1148 1168	Гаріка Корогодського
T3	MISC 1200 1250	Глобальному договору та Цілям сталого розвитку ООН
T4	ORG 1289 1301	Збройних Сил
T5	ORG 1358 1378	Збройні Сили України
T6	LOC 1399 1404	Криму
T7	LOC 1440 1447	Донбасі
T8	ORG 1996 1999	ЗСУ
T9	LOC 2103 2111	Україною
T10	ORG 2144 2158	Збройним Силам
T11	LOC 2318 2325	України
T12	LOC 2371 2376	Росії
T13	ORG 2528 2534	Roshen
T14	LOC 2611 2617	Києвом
T15	ORG 2729 2732	ЗСУ
T16	ORG 2888 2894	Фармак
T17	MISC 2931 2934	КСВ
T18	LOC 3248 3257	Німеччини
T19	LOC 3260 3269	Австралії
T20	LOC 3272 3278	Швеції
T21	LOC 3281 3284	США
T22	ORG 3298 3304	Фармак
T23	LOC 3348 3354	Європи
T24	LOC 3357 3361	Азії
T25	ORG 3506 3512	Фармак
T26	ORG 3542 3550	Екошкола
```

Рисунок 2.2 – приклад вмісту файлу з анотаціями

- оброблені та підготовлені для використанні дані знаходяться в наступній директорії «data/preproc» відносно кореня проєкту в файлі «preproc-data.bin». Даний файл містить об’єкт типу pandas.DataFrame, який має табличний вигляд з наступною структуру: ім’я файлу, список слів в реченні, анотації для кожного зі слів. Приклад вмісту цього файлу можна побачити на рисунку 2.3;

	filename	text	tags
0	A_Halytskyi_korespondent_Fedoliak_Baseyn_peret...	[Не, встигла, новостворена, П'ядицька, ОТГ, за...	[O, O, O, LOC, LOC, O, O, O, O, O, O, O, O,...
1	A_Halytskyi_korespondent_Fedoliak_Baseyn_peret...	[Як, наслідок, ,, на, коломийському, стадіоні,...	[O, O, O, O, O, O, O, ORG, O, O, O, O, O, O, O...
2	A_Halytskyi_korespondent_Fedoliak_Baseyn_peret...	[Турці, .]	[LOC, O]
3	A_Halytskyi_korespondent_Fedoliak_Baseyn_peret...	[Мовляв, ,, тренерам, повідомили, про, звільне...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...
4	A_Halytskyi_korespondent_Fedoliak_Baseyn_peret...	[Присутні, створили, ініціативну, групу, ,, об...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...
...	...	...	...
12009	I_Ivanychuk_1_Torhovytsia_2013(5)	[На, початку, Личаківської, звернув, у, Круняр...	[O, O, LOC, O, O, LOC, LOC, O, O, O, O, O, O, ...
12010	I_Ivanychuk_1_Torhovytsia_2013(5)	[Юрків, родич, ,, член, ОУН, ,, день, відо, дн...	[PERS, O, O, O, ORG, O, O, O, O, O, O, O, O, O...
12011	I_Ivanychuk_1_Torhovytsia_2013(5)	[Примістив, його, в, потаємній, комірці, ,, де...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...
12012	I_Ivanychuk_1_Torhovytsia_2013(5)	[Професор, Сербин, приносив, щораз, тривожніші...	[O, PERS, O, O, O, O, O]
12013	I_Ivanychuk_1_Torhovytsia_2013(5)	[Мало, не, щодня, партприкріплені, до, міських...	[O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...

Рисунок 2.3 – вміст обробленого файлу «preproc-data.bin»

- вхідні конфігураційні дані, гіперпараметри моделі, шляхи до ключових директорій проєкту знаходяться в наступній директорії «configs» відносно кореня проєкту в файлі «config.yml». Даний файл має вигляд, як це зображено на рисунку 2.4.

```yaml
data:
  path_to_corpus_folder: data/corpus
  path_to_preproc_data_folder: data/preproc-data
  path_to_output_folder: data/path_to_output
  path_to_preproc_data: data/preproc-data/preproc-data.bin
  path_to_output: data/path_to_output/output.bin
  path_to_logdir: logdir

model:
  model_name: youscan/ukr-roberta-base
  max_seq_length: 64
  lstm_dim: 256
  lstm_num_layers: 2
  lstm_dropout_rate: 0.3
  lstm_bidirectional_flag: 1
  cnn_dropout_rate: 0.4
  fc_dropout_rate: 0.4
  use_lstm_flag: 0
  use_cnn_flag: 0

training:
  num_epochs: 5
  learning_rate: 3e-3
  accum_steps: 1
  train_batch_size: 32
  valid_batch_size: 8
  log_dir: logdir
  fp16_params: None
  num_workers: 4
  focal_loss_gamma: 0.25
  is_deterministic: 1

general:
  device: cpu # auto, cpu, cuda:0
  seed: 43
  test_size: 0.2
```

Рисунок 2.4 – вміст файлу config.yml

2.2 Вихідні дані

Вихідними даними даного програмного продукту є:

- розмічений текст. Це введений користувачем, під час роботи сервісу, текст, який пройшов процес токенизації та класифікації. Він має вигляд, як зображено на рисунку 2.5;

Рисунок 2.5 – приклад розміченого сервісом тексту

- результуючі, натреновані на навчальній вибірці текстів, параметри моделі записані в файлі «best.pth», що знаходиться в директорії «logdir» відносно кореня проєкту. Збереження даних ваг моделі необхідне для подальшого перевикористання моделі, щоб не

перезапускати процес навчання, який може йти як годинами, так і місяцями, в залежності від кількості даних та архітектури моделі. Завантаження ваг з диску відбувається значно швидше, ніж процес навчання, що зменшить час розгортання моделі та величину потрібних обчислювальних ресурсів для роботи системи. Дані ваги мають структуру багатовимірних тензорів.

2.3 Опис структури бази даних

Для збереження результатів локального запуску сервісу, було прийнято рішення використовувати СКБД SQLite [13]. SQLite є кросплатформною, підтримує доволі широкий набір команд SQL. Її головними плюсами є:

- простота використання;
- можливість локального розгортання;
- вбудована підтримка мовою програмування Python.

Схема БД буде складатись лише з однієї таблиці, куди будуть записувань логи запуску сервісу. Дана схема наведена на рисунку 2.6.

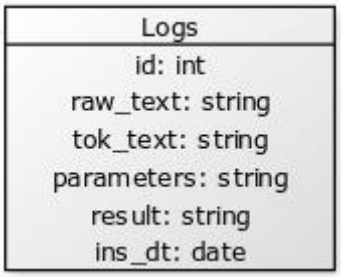


Рисунок 2.6 – схема БД

Детальний опис атрибутів наведено в таблиці 2.1.

Таблиця 2.1 – детальний опис атрибутів таблиці БД

Атрибут	Пояснення	Тип
id	унікальний ідентифікатор запису	int
raw_text	текст, введений користувач у відповідному полі	string
tok_text	токенизований текст	string
parameters	параметри моделі	string
result	результат роботи алгоритму розпізнавання сутностей	string
ins_dt	дата вставки запису	date

2.4 Структура масивів інформації

Для виконання етапу підготовки даних та запуску процесу навчання, необхідно зчитати наявні текстові документи у відповідному каталозі та анотації, що відповідають заданим текстам.

Процес зчитування текстів та анотацій відбувається послідовно. Файли з текстами та анотаціями мають спільну назву, але різне розширення файлів. Так, файли з текстами мають розширення «tok.txt», а файли з анотаціями «tok.ann». Спочатку, зчитуються всі файли, видаляється розширення, а потім залишаються лише унікальні назви файлів. Далі, послідовно, для кожної назви зчитується текст та анотація. Для кожного зчитаного тексту та анотації відбувається розбивка на окремі речення. Кожне слово в реченні отримує певний клас. Якщо слово присутнє в анотаціях, то воно отримує клас з заданого файлу. Якщо слово відсутнє в файлі з анотаціями, то йому ставиться у відповідність клас, що позначає відсутність сутності. Надалі, оброблені дані зберігаються в табличну структуру pandas.DataFrame та зберігаються на дисковий простір. Дану структуру можна побачити на рисунку 2.3.

Ваги натренованої моделі зберігаються в файлі «best.pth», що знаходиться в директорії «logdir» відносно кореня проєкту. Дані параметри моделі мають структуру багатовимірних тензорів, що містять в собі додаткові параметри. Такими параметрами є: градієнти, пристрій на якому відбувається обрахунок і т.д.

Висновок до розділу

В даному розділі було наведено та описано структуру всіх вхідних та вихідних даних, вказано їх особливості та аргументовано їх необхідність. Також, описано структуру БД, що складається з таблиці логів, та всі відповідні атрибути.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

На вхід потрапляє текст, введений користувачем, з певної доменної області. Даний текст може бути заготовлений в окремих файлах чи бути введеним користувачем в спеціальній формі. Після чого, над даним текстом виконується певна обробка та розбиття на токени. Далі, токени переводяться в простір індексів та відправляються на вхід побудованої моделі. Дана модель вирішує задачу класифікації токенів, в даному випадку – це задача розпізнавання іменованих сутностей в текстах. Результатом роботи моделі є тензор наступного розміру:

$$N_{batch} \times N_{\text{maxlen}} \times N_{tags} \quad (3.1)$$

Де N_{batch} – розмір батча, $,N_{\text{maxlen}}$ – максимальна довжина послідовності, N_{tags} – кількість класів в задачі класифікації. В даному тензорі містяться logits ймовірності належності певного токену заданого тексту до одного з класів задачі класифікації – іменованих сутностей. Зрештою, обирається сутність з найбільшою ймовірністю для заданого токену. В результаті, користувач отримує токенізований текст з розміткою, яку вдалось визначити, використовуючи побудовану модель.

Процес навчання відбувається на корпусах текстів, що належать до різних доменних областей: художня література, інтернет видання, новини і т.д.

Дані корпуси текстів проходять попередню обробку, розбиття на слова, примітивне очищення текстів та співставлення анотацій заданим словам. Після чого відбувається розбиття отриманих оброблених текстів на дві вибірки: тестову та навчальну. В подальшому, на навчальній вибірці відбувається навчання нейронної мережі, а тестову вибірку використовують для валідації та оцінки отриманих результатів.

3.2 Математична постановка задачі

Дано: набір текстів з розміченими іменованими сутностями.

Знайти: значення ваг моделі глибокої нейронної мережі, при яких отримується найкраща точність класифікації токенів.

Для пошуку значень ваг потрібно задати диференційовану функцію витрат. В даному випадку, була обрана функція витрат Focal Loss [14], що є загальним випадком поширеної функції витрат Cross-Entropy Loss [15]. Функція витрат Focal Loss чудово підходить для задач з дисбалансом класів та має гіперпараметри для подальшого підбору (рисунок 3.1).

$$FL(p_t) = -(1-p_t)^y \log(p_t) \quad (3.2)$$

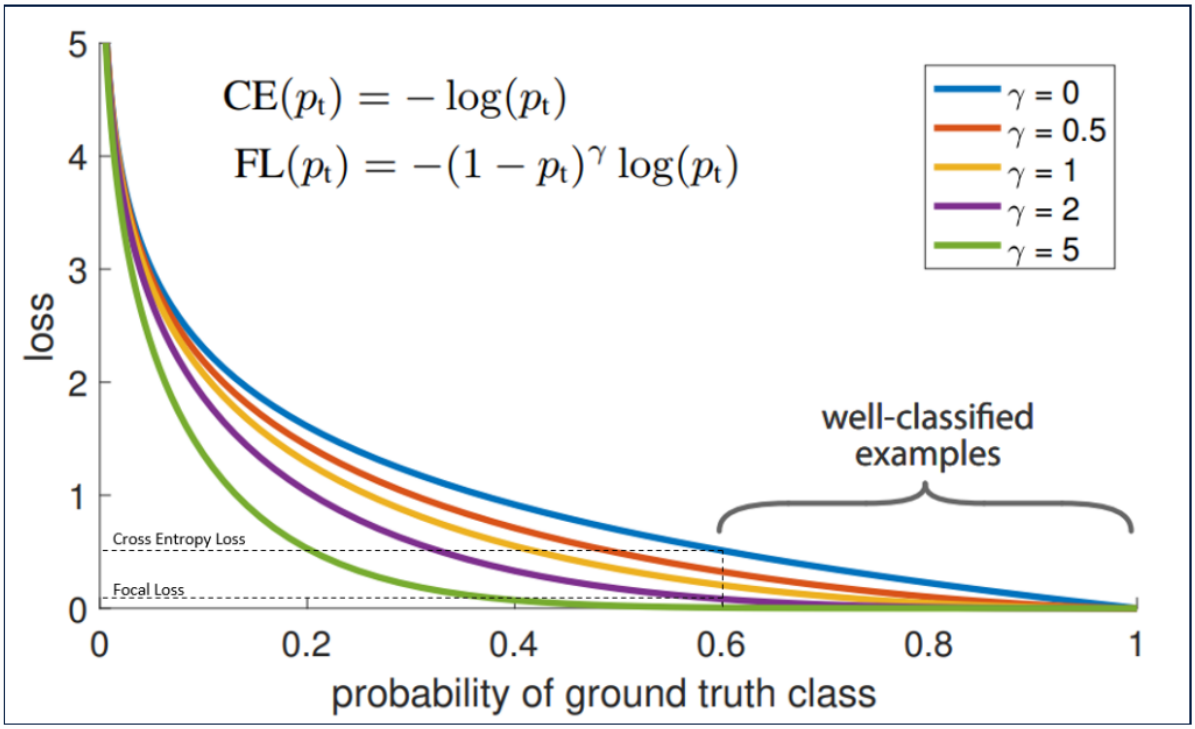


Рисунок 3.1 – порівняння графіків функцій Focal Loss та Cross-Entropy Loss

Введемо позначення logits-ймовірностей p_t для об'єкта x_t, що будуть результатом роботи моделі нейронної мережі:

$$p_t = model(x_t, w) \quad (3.3)$$

Також, w^* - шукані оптимальні значення ваг моделі, x_t – векторне представлення тексту t, y – параметр функції Focal Loss, w – початкові ваги моделі, FL – функція витрат Focal Loss.

Отже, зрештою ми вирішуємо наступну задачу оптимізації:

$$w^* = argmin_w\, FL(p_t) = \quad (3.4)$$

3.1 Обґрунтування методу розв'язання

За основу, для вирішення поставленою задачі, було обрано підхід з використанням нейронних мереж та мовних моделей. Окрім цього, існує велика кількість інших підходів: на основі набору правил, з використанням рекурентних нейронних мереж, тощо.

3.1.1 Обґрунтування вибору нейронної мережі

Перцептрон – математична модель, що була запропонована Френком Розенблатом в 1957 році [16]. Дана математична модель стала одиницею нейронної мережі.

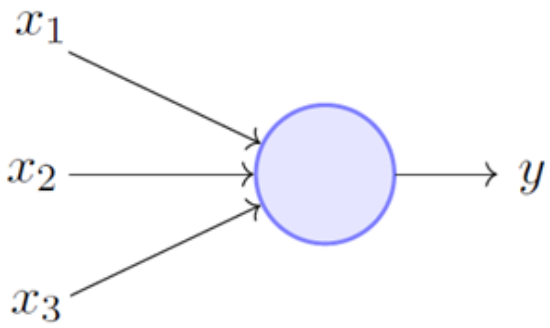


Рисунок 3.2 – модель перцептрона

Дана модель працює досить просто. На вхід подаються певні значення, в даному випадку це параметри x_1, x_2, x_3. На виході маємо певне число y, що приймає значення 0 або 1. Розглянемо детальніше, як саме отримується значення y.

Рисунок 3.3 – модель перцептрона з вагами

Для знаходження результату роботи перцептрона, було запропоновано використовувати певні задані ваги. Дані параметри зображені на рисунку 3.3 як w_1, w_2, w_3. Ці параметри є дійсними числами та можуть бути інтерпретовані як важливість того чи іншого зв'язку, або вплив цього зв'язку на результат роботи моделі. Саме ж число y знаходиться як порівняння зваженої суми вхідних значень $\sum_j x_j w_j$ з певним заданим порогом $threshhold$, що в залежності від задачі може змінюватись.

Даний виразу можна записати у наступному вигляді:

$$y = \begin{cases} 0, \text{якщо } \sum_j x_j w_j \leq threshhlod \\ 1, \text{якщо } \sum_j x_j w_j > threshhlod \end{cases} \quad (3.5)$$

Дана модель є досить простою та може бути реалізована на пристроях з незначною кількістю ресурсів, але вона не може вирішувати складні поставлені задачі, такі як обробка природньої мови. Перцептрон будувався схожим чином до роботи одного нейрона головного мозку. Але людський мозок аналізує велику кількість фактор, перш ніж прийняти рішення. Тому першим логічним кроком в модернізації архітектури прерцептрона є поєднання результатів роботи різних нейронів між собою. Отримаємо рішення, яке зображено на рисунку 3.4 [27].

Рисунок 3.4 – повнозв’язна нейронна мережа

Дане рішення називається повнозв’язною нейронною мережею, яка фактично складається з перцептронів. Архітектурно, дана мережа складається з 3 шарів:

- вхідний шар;
- прихований шар;
- вихідний шар.

Кожен з цих шарів виконує певну функцію. На вхідному шарі дані потрапляють до нейронної мережі. Після чого вони проходять через певну кількість прихованих шарів, основна задача яких – це побудова інформативного представлення даних. Зрештою, вихідний шар має відображати бажаний результат роботи моделі. В залежності від типу задачі цей результат може бути дійсним числом чи ймовірністю.

Варто зауважити, що саме прихований шар є одним з найважливіших шарів нейронної мережі. Існує велика кількість різноманітних варіацій архітектури прихованого шару. Модифікуємо функцію перцептрона, зробивши параметр $threshold$, таким, який буде оптимізуватись в ході навчання нейронної мережі.

$$y = \begin{cases} 0, \text{якщо } \sum_j x_j w_j + bias \ \leq 0 \\ 1, \text{якщо } \sum_j x_j w_j + bias > 0 \end{cases} \quad (3.6)$$

Фактично, замінивши параметр $threshold$, було отримано параметр $bias$, який може бути інтерпретований як значення, якого має набувати зважена сума, щоб змінна y набувала значення 1.

Отже, ми отримали набір параметрів, оптимізуючи які, нейронна мережа буде навчатись приймати ті чи інші рішення на основі вхідних даних.

Але, існує проблема в такому підході. У випадку зміни вхідних параметрів на незначну величину, може відбутись зміна значення параметру з 0 на 1, або ж навпаки. Цю проблему вирішує підхід, що пропонує застосовувати функції активації.

Також, іншою аргументацію до використання функцій активації слугує факт, що фактично перцептрон є лінійною згорткою, яка передається на вхід іншому перцептрону. Зрештою, ми отримуємо, послідовність лінійних згорток. Розписавши такий вираз, ми бачимо, що наша нейронна мережа, в якій відсутні функції активацій, є одним великим лінійним шаром. Тим самим, вона навчається будувати лише лінійні функції розділення даних і не може враховувати нелінійні належності. В реальному житті, більшість даних має нелінійну природу та не можу бути розділена простою лінійною функцією.

Тому пропонується застосовувати певні функції активацій, які б накладали на зважені суми перцептронів певні нелінійні функції, що дадуть змогу вирішити описані вище проблеми. Прикладами таких функцій можуть бути:

- сигмоїдна функція активації – дані функцію також називають логістичною функцією активаці, що є гладкою монотонно зростаючою нелінійною функцією.

$$\sigma = \frac{1}{1 + e^{-z}} \quad (3.7)$$

Рисунок 3.5 – графік сигмоїдної функції активації

Як ми можемо бачити, дана функція є диференційованою, що є важливим фактом в контексті навчання нейронних мереж. Також, вона може набувати значень від 0 до 1, що може бути інтерпретовано як ймовірність настанні тієї чи іншої події. Але, дана функція має певні недоліки в контексті навчання нейронних мереж. Основними підходами до навчання нейронних мереж є використання градієнтних методів. А отже, ми маємо обраховувати значення похідної від функції активації при зміні ваг моделі. Враховуючи, що більшість нейронних мереж є досить глибокими, при обрахунку градієнтів за правилом ланцюга, ми отримаємо відносно невелике значення градієнта. Тим самим, наші ваги майже не будуть змінюватись і процес навчання не буде відбуватись. Така проблема називається проблемою затухаючого градієнта. Також, подібні функції потребують значних ресурсів системи для обрахунку;

− гіперболічний тангенс – дана функціє є скоректованою версією логістичної функції активації. Вона має ті самі плюси та мінуси, що і логістична функція, але набуває значень в діапазоні від -1 до 1. Також, враховуючи діапазон значень, яких може набувати функція, при розрахунку значень знімається обмеження на додатне значення функції активації. Також, значення похідної в околі нуля в даної функції значно більше, що дає

можливість нейронній мережі швидше навчатись та сходитись до оптимального значення;

$$\tanh(z) = 2sigmoid(2z) - 1 \quad (3.8)$$

Рисунок 3.6 – графік функції активації гіперболічного тангенсу

– ReLU – одна з найпопулярніших функцій активації. Дана функція повертає 0, якщо аргумент менше нуля, в разі додатного аргументу повертається саме ж значення аргументу. Головним плюсом використання даної функції є швидкість обрахунку значень функці, швидкість обрахунку значень похідної, необмежене зверху значення похідної. Недоліком ж є нульове значення похідної при від'ємному значенні аргументу;

$$f(z) = \max(0, z) \quad (3.9)$$

Рисунок 3.7 – графік функції активації ReLU

– Leaky ReLU – як вже було сказано, основною проблемою використання функції ReLU є нульове значення похідної при від'ємному значенні аргументу. Дана функція активації вирішує цю проблему. Основними недоліками використання даної функції є: ускладнення обчислень, необхідність в підборі кутового коефіцієнту, на практиці даний підхід не завжди покращує результат роботи нейронної мережі.

$$f(x) = \begin{cases} 0.01x, & \text{якщо } x < 0 \\ x, & \text{якщо } x \geq 0 \end{cases} \quad (3.10)$$

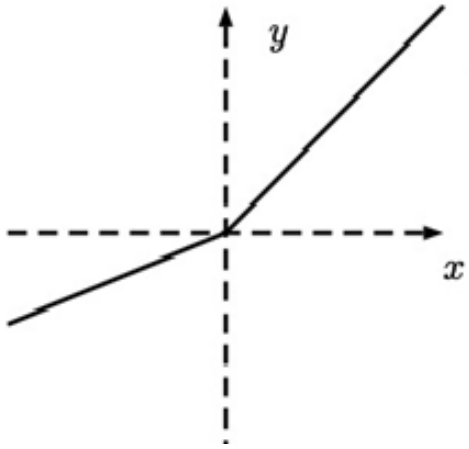


Рисунок 3.8 – графік функції активації Leaky ReLU

3.1.2 Обґрунтування вибору функції витрат

Правильний вибір функції витрат є одним з найголовніших етапів вирішення задачі оптимізації. В даному випадку ми маємо задачу багатокласової класифікації. Розглянемо можливі функції витрат, що могли б допомогти вирішити поставлену задачу:

– Cross-Entropy Loss – дана функція витрат є загальним випадком функції Binary Cross-Entropy. На вхід цієї функції подаються logits ймовірності. Вона може вирішувати задачу багатокласової класифікації.

$$CE = -\sum_{c=1} y_c \log(p_c) \quad (3.11)$$

Де в формулі 3.11 y_c – істинний клас, p_c – logits ймовірність.

- Focal Loss – дана функція витрат є узагальненим випадком Cross Entropy Loss та за рахунок можливості зміни свої гіперпараметрів може бути використана для вирішення задач з дисбалансом класів. Відповідний графік даної функції можна побачити на рисунку 3.1.

3.1.3 Обґрунтування вибору базової архітектури нейронної мережі

Для вирішення поставленої задачі класифікаї токенів може бути використано декілька різних підходів, але ключовою особливістю кожного з них є побудова інформативного векторного представлення слів тексту. Даний етап є необхідним, оскільки алгоритми вміють працювати виключно з числовими даними.

Побудувати інформативне векторне представлення можна декількома способами:

- Використовуючи статистичні особливості текстів. Одним з таким підходів є TF-IDF [17]. Даний підхід дозволяє оцінити важливість того чи іншого терміну для якогось документу відносно всіх інших. Дана величина складається з двох основних компонент: частота терміна, яка показує наскільки часто зустрічається слово в документі, та обернена частота документів в яких зустрічається термін. Її ключові клюси: легко обраховується та неважливі для всіх документів слова отримують низьке значення цієї величини. Але сама по собі метрека є статистичною та не враховує контекст, в якому зустрічається те чи інше слово.
- Іншим підходом до побудови інформативного векторного представлення слів в тексті є використання алгоритму Word2Vec. Даний алгоритм навчається не великих корпусах текстів та вирішує задачу прогнозування слова по контексту, в якому воно зустрічається. В ході вирішення поставленої задачі формується представлення слів

в векторному вигляді заданого розміру. Дане представлення називається ембедінгом слова. В залежності від розміру контексту, його ще називають вікном, алгоритм буде охоплювати певну кількість слів. Даний процес зображено на рисунку 3.9. Для покращення результатів роботи даної моделі додатково застосовується певні техніки оптимізації, підбору негативних прикладів та варіюються гіперпараметри задачі.

Рисунок 3.9 – процес підготовки даних для алгоритму Word2Vec

- Використовуючи моделі, що формують векторне представлення слів в залежності від поданого на вхід контексту. Однією з таких моделей є модель BERT. Її ключова особливість в тому, що під час навчання вирішувалось одразу декілька задач оптимізації: прогнозування замаскованого слова та класифікації чи є задане речення продовженням іншого. Дана модель має дуже багато параметрів та використовує останні SOTA підходи до роботи з даними, які мають послідовну структуру. Подібні моделі навчаються досить довго та потребують значної кількості обчислювальних ресурсів. Тому, використовують вже навчені моделі з інформативними ембедінгами. В разу, якщо потрібно застосувати дану модель для специфічної

області, то ваги «заморожують», тим самим забороняючи через них протікати градієнту. Це призводить до того, що параметри моделі BERT не будуть змінюватись. Тоді ж як модель буде вчитись? Для цього результат моделі BERT буде потрапляти на вхід наступним шарам нейронної мережі, через які вже зможе протікати градієнт та оновлювати параметри моделі. Сам BERT за архітектурою складається лише з енкодера, який за своєю структурою схожий на звичайний енкодер з архітектури трансформера.

3.1.4 Обґрунтування вибору архітектури останнього шару моделі

Даний етап є досить важливим, оскільки саме даний шар архітектури буде вирішувати як саме використовувати отримані інформативні векторні представлення. На даному етапі проводився вибір одного з наступних варіантів архітектури:

- застосування лінійного шару до кожного з отриманих векторних представлень слів – даний спосіб є дієвим в тому випадку, коли побудовані векторні представлення вже містять інформацію про контекст, в якому зустрічаються ці слова. В іншому ж випадку, ми маємо враховувати які слова стоять поруч.
- застосування рекурентного шару – в даному випадку мова йде про застосування блоку LSTM [19]. Основним завданням даного типу блоку є зберігання інформації впродовж довгого періоду часу. Блок Long Short-Term Memory містить ключовий компонент cell state, що дозволяє пропускати через себе інформацію про попередні стани. Також, даний тип блоку містить певні фільтри (gates), що на підставі значень функції активації вирішують пропускати дану інформацію чи ні. Спочатку, LSTM вирішує яку інформацію ми збираємось викинути зі стану, використовуючи логістичну функцію активації. На наступному етапі вирішується яку інформацію потрібно зберегти в

cell state. Даний процес складається з двох етапів. По-перше, логістичний шар вирішує які значення треба оновити. Потім, вже шар з функцією активації гіперболічного тангенсу створює вектор нових значень, які додає в стан. Тепер попередній стан комірки може бути оновлений, базуючись на отриманих вище значеннях. Опис архітектури даного блоку наведений на рисунку 3.10 [28]. Також, подібні блоки можуть об'єднуватись в окремі шари, як це показано на рисунку 3.11.

Рисунок 3.10 – архітектура LSTM блоку

Рисунок 3.11 – два шари з блоків LSTM

3.2 Опис методів розв'язання

Базуючись на методах та підходах до побудови архітектури та особливостей використання, було обрано підхід, що дасть оптимальні результати з точки зору швидкості роботи та якості прогнозування.

3.2.1 Особливості підготовки даних для навчання

Як вже було описано, що моделі вміють працювати лише з числовими представленнями даних, аж ніяк не словами чи сирими текстами. А вхідні дані в нас представлені як файли з сирими текстами та файли з анотаціями. Анотаціями в рамках цієї задачі виступають іменовані сутності.

Всього, вхідний корпус текстів налічує 12014 речень, 264 тексти, 241383 слів та 50293 унікальних слів. Середня довжина речення складає 20 слів, а медіана складає 16 слів.

Першим етапом обробки стало розбиття текстів на окремі токени та співставлення кожному токену певного класу. В даній задачі розпізнавання іменованих сутностей маємо наступні класи:

- LOC (локації) – назви міст, регіонів, тощо;
- PERS (персони) – імена, прізвище, клички, тощо;
- ORG (організації) – назви організацій, підприємств, тощо;
- MISC (різноманітні сутності) – назви продуктів, категорій, тощо.

Зрештою, оброблені дані були збережені на диску для подальшого використання. Структура цього файлу зображена на рисунку 2.3.

Але наші дані все ще представлені в текстовому вигляді, аж ніяк не числовому. Найпростішим варіантом вирішення цієї проблеми є створення словника, який буде ставити у відповідність певному слову заданий індекс. В подальшому, базуючись на числовому індексі модель буде повертати те чи інше значення інформативного векторного представлення заданого розміру. У такого підходу існує ряд суттєвих недоліків:

- що робити у разі відсутності слова в словнику?

- що робити з різними формами слів?
- що робити зі словами, в яких допущені помилки при написанні?
- як враховувати суфікси, префікси та закінчення слів, які можуть нести інформацію про контекст, в якому вживається слово?

Дійсно, існують сутності та підходи, які дозволяють приводити слова до нормальних форм, відкидаючи закінчення та суфікси, але в такий спосіб ми будемо втрачати частину інформації.

Також залишається відкритою проблема з невідомими словами. Можна робити їх заміну на спеціальний індекс, який буде відповідати за невідомі в тексті слова, але в такий спосіб ми втрачаємо інформацію про контекст. Також, до невідомих слів будуть віднесені всі слова, в яких зустрічаються помилки в написанні. Хоча насправді, дані слова можуть бути важливими та суттєвими для розуміння того, що відбувається в реченні.

Іншою проблемою є те, що ми просто не зможемо зібрати набір текстів достатнього об’єму, аби побудувати словник всіх можливих слів. При розпізнаванні іменованих сутностей це дуже важливо, оскільки ми не зможемо правильно розпізнати сутність, якої немає в словнику.

Вирішити поставлену задачу допоможе підхід до токенизації, що називається Word Piece. Даний алгоритм токенизації є токенизацією по підсловам, що використовується в моделях BERT, DistilBERT, тощо. Спочатку, даний алгоритм виконує ініціалізацію словника, в який включає всі можливі символи з заданого тексту та поступово вивчає наступні комбінації цих символів, використовуючи певні правила. Даний алгоритм максимізує ймовірність того що дані будуть додані в словник. Вже після чого, кожному запису зі словника у відповідність буде поставлено певний індекс. Приклад такої токенизації зображено на рисунку 3.12.

Рисунок 3.12 – приклад Word Piece токенизації

Залишається вирішити ще одну проблему. Моделі, такі які нейронні мережі, вміють працювати з фіксованою кількістю вхідних параметрів, але кожне речення має різну довжину. Пропонується ввести гіперапарметр, який буде відповідати за максимально можливу довжину речення. В разі, якщо довжина речення менше максимально можливої, тоді відбувається доповнення речення спеціальним токеном. По даному токену не відбувається обрахунок функції втрат та він ніяк не впливає на значення метрик валідації.

3.2.2 Обрана архітектура нейронної мережі

Для реалізації поставлених цілей та розробки моделі нейронної мережі було використано фреймворк з відкритим вихідним кодом PyTorch [20]. Основним плюсом даного фреймворку є його гнучкість та побудова динамічного графа обчислень, на відміну від інших пакетів, які будуть статичні графи. Даний фреймворк може працювати як на CPU, так і на GPU, що значно прискорює виконання матричних операцій.

За основу для побудови інформативних векторних представлень використовується бібліотека Transformers від HuggingFace [21]. Дана бібліотека містить набір необхідних сутностей для побудови моделі BERT.

Оскільки вирішення поставленої задачі має відбуватись українською мовою, використовується навчена модель ukr-roberta-base [22]. Дана модель навчалась корпусах українських текстів відповідно до рекомендацій HuggungFace. Також, ця модель має наступну архітектуру:

- 12 шарів;
- прихований шар розміром 768 ;
- 12 голів;
- 125 мільйонів параметрів.

Тепер, поглянемо на саму архітектуру моделей BERT детальніше та розберемо їх процес навчання. Як вже було описано, для початку відбувається збір великої кількості даних та їх попередня обробка. В якості токенізатора використовується Word Piece алгоритм.

Навчання даної моделі є доволі складним процесом, оскільки має бути здійснена велика кількість математичних операцій. В ході навчання вирішується одразу декілька задач оптимізації:

- задача прогнозування замаскованого слова – для цього штучно виділяють 15% слів в тексті довільним чином та замінюють їх на спеціальний токен, який позначає невідоме слово. В результаті, нейронна мережа має спрогнозувати коректне слово, яке має стояти на місці цього токена.
- задача класифікації – з заданого тексту обираються довільно 2 фрази і вирішується задача прогнозування чи є перше речення продовженням другого.

Зрештою, після завершення навчання, модель BERT навчиться досить добре розуміти контекст та будувати інформативні векторні представлення.

Модель BERT-base складається з 12 шарів енкодерів, які ще називають блоками трансформерами.

Рисунок 3.13 – шари моделі BERT-base

Як вже було сказано, над текстом відбувається токенизація. Також, туди додаються технічні токени, такі як: токен класифікації, токен розділювач, тощо. Першим знаходиться саме токен класифікації, як це зображено на рисунку 3.14.

Рисунок 3.14 – вхід моделі BERT

Модель BERT приймає на вхід набір індексів і передає на вхід першому шару. Результат роботи першого шару проходить через повозв'язну нейронну мережу і передається на вхід іншому шару, як це зображено на рисунку 3.15.

Рисунок 3.15 – процес послідовного розповсюдження результати в моделі BERT

На виході даної моделі ми отримаємо для кожного токена своє інформативне векторне представлення заданого розміру, як це показано на рисунку 3.16.

Рисунок 3.16 – отримання векторного представлення слів

В подальшому, отримані вектори можуть бути використані для вирішення задачі класифікації, використовуючи підхід fine-tuning.

3.2.3 Особливості етапу навчання нейронної мережі

Перш за все, варто зауважити, що для навчання нейронної мережі використовувався підхід, що називається fine-tuning. Для реалізації цього підходу, необхідно мати нейронну мережу, що вміє дуже добре вирішувати певну поставлену задачу. В нашому випадку, ми вирішуємо задачу розпізнавання іменованих сутностей, що є однією з задач NLP. Для отримання якісного результату, модель має будувати інформативні векторні представлення текстів. Тому, для вирішення даної задачі була обрана модель BERT, що є SOTA підходом до вирішення задач NLP. Всі ваги даної моделі були заморожені, тобто для них примусово було вимкнено обрахунок градієнту. Тоді виникає питання, а що ж буде навчатись? Для цього, вихід моделі BERT потрапляє на ще один шар. В даному випадку це звичайний лінійний шар або LSTM. Саме ваги цього шару і будуть оновлюватись. Тим самим, ми отримаємо ситуацію, коли модель BERT будує інформативні векторні представлення слів, базуючись на всій послідовності наданих слів та їх порядку, а в подальшому інший шар моделі приймає рішення про те до якої сутності належить те чи інше слово. Також, даний підхід прискорює навчання, зменшує об’єм необхідних ресурсів для зберігання моделі та допомагає краще уникати перенавчання. Але в нього є і свій недолік: модель BERT навчалась на інших даних, які можуть бути зовсім не пов’язані з доменною областю, що погіршить фінальний результат роботи нейронної мережі.

Також, для навчання було використано модифікацію градієнтного спуску AdamW [23] та техніку зменшення швидкості навчання Linear Schedule with Warmup [24]. Використання даних технік оптимізації дозволяють краще та швидше сходитись до бажаного результату.

Результуючий код моделі був обгорнутий в фреймворк Catalyst [25], завдяки якому вдалось отримати відтворювані результати експериментів та уникнути написання довгих шматків коду та циклів.

3.2.4 Результати навчання нейронної мережі

Під час процесу навчання збирались метрики, які обраховувались на навчальній та валідаційній вибірці.

На рисунку 3.17 зображено графік цільової функції втрат, що оптимізувалась в ході вирішення поставленої задачі. По осі вертикальній осі відкладено значення функції втрат, а по горизонтальній відкладено кількість пройдених ітерацій. Також, оранжевим кольором виділено значення метрики, що обраховувалось на навчальній вибірці, а синім значення, що обраховувались на валідаційній вибірці. Варто зауважити, що оранжевий графік знаходиться над синім графіком – це говорить про те, що модель не перенавчилась в ході процесу тренування.

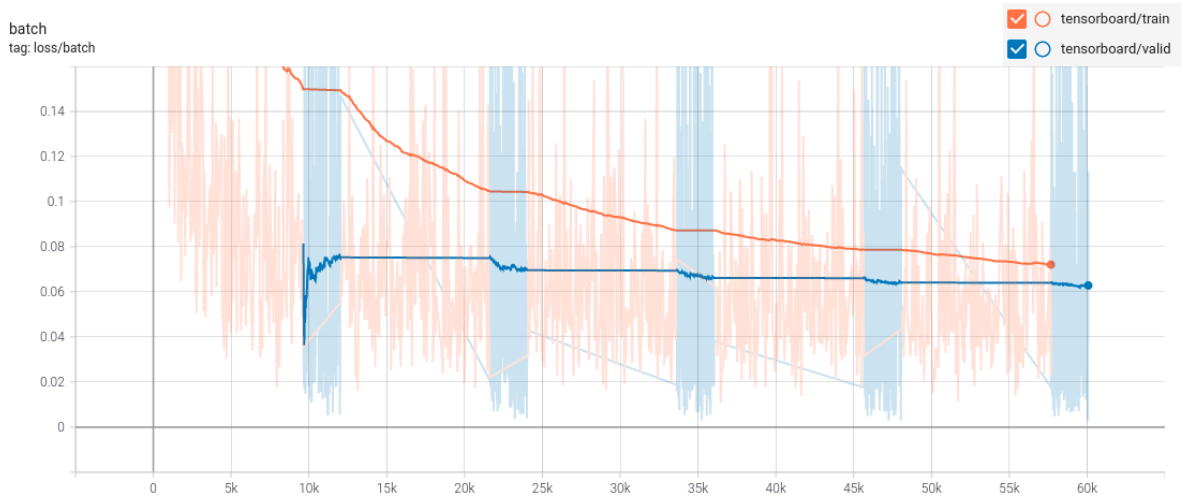


Рисунок 3.17 – графік функції втрат

На рисунку 3.18 зображено графік точності класифікації. По вертикальній осі відкладено значення точності класифікації, а по горизонтальній осі відкладено кількість пройдених ітерацій. Значення даної метрики становить приблизно 96.9% на тестовій вибірці.

Рисунок 3.17 – графік точності класифікації

При обрахунку вказаних вище метрик, ігнорувались всі допоміжні токени, як такі, що не впливають на результат.

Висновок до розділу

В даному розділі математичного забезпечення було сформовано змістовну та математичну постановку задачі, вказано цільову функцію оптимізації. Також, описано причину вибору даного способу розв’язання, його переваги та недоліки. Було наведено і інші способи та підходи до розв’язання. Також, було детально розібрано принципи роботи моделі BERT та деяких шарів нейронної мережі. Зрештою, було обрано архітектуру моделі, підхід до розв'язання, вказано ключові особливості до підходу навчання, вказано обрані техніки оптимізації. Вкінці, наведено результат оптимізації нейронної мережі, вказано значення метрик, зібраних на валідаційній та навчальній вибірках.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для реалізації даного проєкту було обрано мову програмування Python, оскільки дана мова є високорівневою та для неї написана велика кількість модулів і бібліотек, які можуть вирішувати різноманітні задачі оптимізації. Також, дана мова є інтерпретованою, що значно полегшуєш написання та тестування коду. Варто додати, що код мовою програмування Python можна писати як в функціональному стилі, так і об'єктно-орієнтованому, тим самим беручи найкраще з цих двох підходів.

Мова програмування Python є основним інструментом для розробки нейронних мереж. В ній присутні всі найпопулярніші бібліотеки з цієї сфери, де реалізовані функції та класи для роботи з нейронними мережами: Pytorch, Keras, TensorFlow, Theano, тощо. Також, присутня велика кількість бібліотек, що полегшують аналіз, обробку та різноманітні маніпуляції з даними.

Реалізація даного програмного забезпечення відбувалась мовою програмування Python 3.8. Були використані наступні бібліотеки:

- pytorch – сучасний фреймворк, що містить велику кількість функцій та класів для реалізації нейронних мереж та допоміжних сутностей. Особливістю цього фреймворка є велика гнучкість, можливість роботи з GPU та побудова динамічного графу обчислень, на відміну від інших фреймворків, які будують статичний граф обчислень;
- transformers – бібліотека, реалізована компанією HuggingFace, в якісь міститься велика кількість допоміжних сутностей та навчених моделей типу BERT. Дана компанія створила документацію, згідно з якою кожен може навчити свою мережу для вирішення певних задач та опублікувати це в спеціальному репозиторії. Враховуючи недостатній розвиток NLP для

української мови, існує незначна кількість натренованих BERT моделей. Одну з навчених моделей можна завантажити, використовуючи відповідну бібліотеку;

- catalyst – фреймворк для глибокого машинного навчання, за допомогою якого можна писати зручний код, без надлишкових циклів, та отримувати відтворювані результати;
- sklearn – бібліотека для машинного навчання, в якій міститься велика кількість допоміжних функцій для аналізу, обробки, підготовки даних. Також, в ній реалізовано досить багато класичних моделей та алгоритмів машинного навчання;
- numpy – бібліотека, що дозволяє ефективно працювати з векторними представленнями даних. Незважаючи на те, що Pyhton є динамічною мовою програмування та має певні обмеження на використання багатопотоковості, дана бібліотека є надзвичайно швидкою. Більшість її функцій написані на низькорівневих мовах програмування та знаходяться в скомпільованому вигляді, що пришвидшує обробку масивів даних;
- pandas – бібліотека, що дозволяє зручно працювати з табличними представленнями даних, представляючи їх як об’єкт мови Python. Дана бібліотека тісно пов’язана з бібліотекою numpy;
- joblib – бібліотека, що містить набір функцій для побудови різноманітних пайплайнів. Також, містить функції для кешування та реалізації паралельних обчислень;
- pyyaml – бібліотека для роботи з даними, збереженими у файлі формату yaml;
- json – бібліотека для роботи з даними, представленими в форматі json;
- os – бібліотека, що реалізує набір функцій для взаємодії з операційною системою;

- streamlit – фреймворк, що дозволяє швидко та ефективно розробляти веб застосунки для візуалізації різноманітних результатів. Даний фреймворк підтримує функції кешування, що значно пришвидшує роботу;
- sqlite3 – вбудована бібліотека, що дозволяє зручно працювати з БД SQLite.

Розробка та прототипування архітектури нейронної мережі відбувались за допомогою інтерактивного середовища розробки Jupyter Notebook, що представлений в пакеті Anaconda. В подальшому, після етапу прототипування, код був перенесений в модульний вигляд. Фінальна версія програмного продукту була реалізована з використанням середовища Visual Studio Code.

Тестування та розробка програмного продукту відбувалась в операційній системі Pop OS 20.04.

Для збереження результату роботи програмного продукту використовувалась СКБД SQLite, оскільки вона може зберігати результат локально, легко розгортається та відповідає всім поставленим вимогам.

4.2 Вимоги до технічного забезпечення

Для коректної роботи даної системи виділено наступні вимоги до технічного забезпечення:

До складу технічних засобів повинні входити:

- комп’ютер з такою конфігурацією:
  1) процесор з тактовою частотою не нижче 2.4 Ггц та не менше ніж 2 ядрами;
  2) 32 або 64 бітна система;
  3) об’єм оперативної пам’яті не менше 8гб;
  4) інші складові можуть мати довільні параметри;
- на комп’ютер має бути встановлено наступне ПО:
  1) Python 3.8;

2) пакетний менеджер poetry [25];
3) встановлено всі бібліотеки, що містяться в конфігураційному файлі pyproject.toml відповідного пакетного менеджера;

- комп'ютерна периферія, до складу якої входять:
  1) монітор;
  2) маніпулятор мишка;
  3) клавіатура.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма компонентів

У графічному матеріалі наведено діаграму компонентів. На даній діаграмі зображено деталі взаємодії користувача на абстрактному рівні.

4.3.2 Діаграма класів

Діаграма розроблених класів наведена в графічному матеріалі.

Кожен з представлених класів відіграє свою роль при навчанні моделі:

- AccuracyCallbackCustom – клас, в якому відбувається обрахунок метрики під назвою «точність класифікації». В подальшому, цей результат використовується для валідації моделі та перевірки коректності її роботи;
- CrossEntropyLossCustom – клас, який являє собою функцію витрат cross-entropy loss. Використовується для оптимізації моделей при вирішенні задач багатокласової класифікації.
- FocalLossCustom - клас, який являє собою функцію витрат focal loss. Використовується для оптимізації моделей при вирішенні задач багатокласової класифікації, з можливістю налаштування гіперпараметрів, що дозволяють даній функції витрат працювати в умовах дисбалансу класів.

- CustomRunner – клас, в який обгортається побудована модель. Це необхідно для отримання відтворюваних результатів експериментів та зручності розгортання моделі.
- NamedEntityRecognitionBertModel – клас, що містить архітектуру побудованої моделі розпізнавання іменованих сутностей, параметри та всі необхідні методи для її використання.
- NamedEntityRecognitionDataset – клас, що містить в собі всі необхідні методи для завантаження даних, обробки, підготовки та передачі їх на вхід моделі.

4.3.3 Діаграма залежності пакетів

Діаграма залежність модулів та пакетів в розробленому програмному забезпеченні наведена в графічному матеріалі. На ній зображено ієрархічну залежність розроблених модулів програмного забезпечення та зв’язки між ними.

4.3.4 Діаграма розгортання

Діаграма розгортання наведена в графічному матеріалі.

Розгортання даної системи може відбуватись на локальній системі. Для роботи розробленого програмного забезпечення необхідно мати встановлений інтерпретатор мови програмування Python.

4.3.5 Специфікація функцій

В таблиці 4.1 наведено специфікацію функцій даного програмного забезпечення.

Таблиця 4.1 – Функції програмного забезпечення

Назва	**Опис**
get_config(filename)	Дана функція повертає словник, який містить вхідні параметри проєкту. Дані параметри зчитуються з відповідного конфігураційного файлу config.yml
get_model()	Дана функція повертає об’єкт моделі нейронної мережі. Параметри даної моделі зчитуються з файлу best.pth, що містить ваги моделі, які відповідають стану, в якому досягалось найменше значення функції витрат.
get_general_page()	Дана функція відповідає за відображення головної сторінки проєкту. В ній відбувається ініціалізація моделі , виконання аналізу тексту та відображення результату.
annotation(body, label, background, color, style)	Дана функція обертає переданий текст в html-тег для подальшого відображення результату на відповідній сторінці.
annotated_text(*args, **kwargs)	Дана функція перетворює переданий набір токенів в html-блок для подальшого відображення.
remove_dir(path)	Функція для видалення вказаної папки зі всіма вкладеними файлами.
get_filenames()	Функція, що виконує пошук файлів з текстом та відповідних йому анотацій.
process_annotation(annotation)	Функція, що виконує виокремлення анотацій та відповідної частини тексту.
raw_text_to_parallel()	Функція, що виконує співставлення кожного слова тексту певній іменованій сутності.
_getitem_encoded(index)	Отримання закодованого тексту за заданим індексом.
_get_item_lazy(index)	Отримання індексів тексту, маски та анотацій відповідних токенів по заданому індексу.

Продовження таблиці 4.1

forward(input_ids, attention_mask, **kwargs)	Функція обрахунку logits-ймовірностей моделі
freeze()	Замороження параметрів моделі BERT.
unfreeze()	Розмороження параметрів моделі BERT.
forward(output, target, attention_mask, output_dim)	Функція для розрахунку значень функціх витрат.
handle_batch(batch)	Функція, в якій виконується один крок навчання моделі.

Висновок до розділу

У даному розділі було описано засоби розробки, модулі та бібліотеки, що використовуються для розробки програмного продукту. Також, описано вимоги до технічного забезпечення, необхідного для розгортання та запуску програмного продукту. Наведено діаграми залежності пакетів, опис розроблених класів та діаграму розгортання програмного продукту. Також, описано специфікацію функцій, в якій наведено призначення відповідної функції та її вхідні дані.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Користувач потрапляє на головну сторінку сервісу розпізнавання іменованих сутностей в тексті.

Рисунок. 5.1 – Головна сторінка сервісу розпізнавання іменованих сутностей

Використовуючи меню зліва, користувач має можливість перейти на головну сторінку, сторінку інформації про навчання моделі та сторінку з історії запусків.

На головній сторінці користувач має можливість ввести текст у відповідне поле або обрати готовий шаблон тексту. Для того, аби програма розпочала роботу над розпізнаванням сутностей для обраного тексту потрібно натиснути на кнопку Submit. У результаті користувач отримає текст з виділеними іменованими сутностями, як це зображено на рисунку 5.2

Рисунок 5.2 – Результат роботи сервісу з розпізнавання сутностей

Перейшовши на сторінку інформації про навчання моделі користувач має змогу переглянути загальні відомості щодо побудови моделі для розпізнавання іменованих сутностей, графік зміни функції витрат у залежності від кількості пройдених ітерацій та графік зміни точності класифікації.

Рисунок 5.3 – Сторінка інформації про навчання моделі

На сторінці з історії запусків користувач може переглянути вхідні параметри моделі, результат виконання алгоритму розпізнавання сутностей та час роботи запуску для кожного переданого тексту.

Рисунок 5.4 – Сторінка з історії запусків

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій сервісу з розпізнавання іменованих сутностей вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Під час виконання мануального тестування сервісу з розпізнавання іменованих сутностей було перевірено весь функціонал. Програмний код

розробленого сервісу знаходиться в Додатку А. Результат даних випробувань представлено у наступних таблицях (5.1 – 5.3).

Таблиця 5.1 – Тестування функції введення тексту у відповідне поле.

Мета тесту	Коректність введення тексту у відповідне поле.
Початковий стан	Відкрита головна сторінка сервісу з розпізнавання іменованих сутностей.
Вхідні дані	Введений текст або обраний шаблон тексту.
Схема проведення тесту	Ввести текст або обрати шаблон тексту та натиснути на кнопку Submit.
Очікуваний результат	Отримати текст з виділеними іменованими сутностями.
Результуючий стан	Отримати текст з виділеними іменованими сутностями.

Таблиця 5.2 – Тестування роботи функціоналу з введення пустого рядочку у відповідне поле

Мета тесту	Введення пустого рядочку у відповідне поле.
Початковий стан	Відкрита головна сторінка сервісу з розпізнавання іменованих сутностей.
Вхідні дані	Пустий текст.
Схема проведення тесту	Натиснути на кнопку Submit.
Очікуваний результат	Отримати повідомлення про необхідність введення тексту.
Результуючий стан	Отримати повідомлення про необхідність введення тексту.

Таблиця 5.3 – Тестування роботи функціоналу з введення максимальної кількості слів у відповідне поле.

Мета тесту	Введення максимальної кількості слів у відповідне поле.
Початковий стан	Відкрита головна сторінка сервісу з розпізнавання іменованих сутностей.

Продовження таблиці 5.3

Вхідні дані	Текст, кількість слів якого перевищує ліміт.
Схема проведення тесту	Ввести достатньо велику кількість тексту та натиснути на кнопку Submit.
Очікуваний результат	Отримати повідомлення про те, що введений текст перевищує встановлений ліміт кількості слів.
Результуючий стан	Отримати повідомлення про те, що введений текст перевищує встановлений ліміт кількості слів.

Висновок до розділу

У даному розділі було описано керівництво користувача, де перелічено які дії може виконувати користувач в розробленій системі, яким чином він має їх виконувати. Також, було описано мету випробувань, загальні положення та наведено мету випробувань даного програмного продукту. Проведено випробування основного функціоналу – розпізнавання іменованих сутностей в текстах. У результаті випробувань було виправлено усі виявлені помилки.

ЗАГАЛЬНІ ВИСНОВКИ

У результаті виконання даного проєкту було розроблено сервіс з розпізнавання іменованих сутностей в текстах для української мови з використанням SOTA мовних моделей BERT.

У розділі з описом предметного середовища було детально описано предметну область, процес діяльності та функціональну модель. Процес діяльності представлений діаграмою діяльності користувача та системи. Було визначено, що із системою буде взаємодіяти лише користувач – особа, що працює з відповідним сервісом. Також, було визначено функціональні вимоги та встановлено пріоритетності для кожної з них. Було проведено пошук застосунків, що мають аналогічну чи схожу функціональність. Більшість з них мають постійну підтримку та розвиваються, але наразі вони не мають якісної підтримки української мови та майже не використовують мовні моделі BERT для побудови інформативних векторних представлень. Також, в цьому розділі було детально описано призначення даного сервісу, встановлено цілі та задачі розробки.

У розділі інформаційного забезпечення було описано вхідні та вихідні дані. Вказано особливості цих даних, призначення, наведено їх структуру. Вхідними даними цього проєкту є корпуси розмічених текстів та ваги нейронної мережі. Вихідними даними є розмічений та токенізований текст, з використанням побудованої моделі розпізнавання іменованих сутностей. Також, в даному проєкті використовується СКБД SQLite для зберігання історії користування побудованим сервісом. В даному розділі було наведено структуру відповідної таблиці зі збереженим результатом.

У розділі математичного забезпечення було сформовано змістовну та математичну постановку задачі, вказано функцію оптимізації, описано існуючі підходи, теоретичну частину та методи, за допомогою яких зрештою відбувалось розв’язання поставленої задачі. В результаті дослідження, було прийнято рішення щодо використання функції Focal Loss в якості функції

оптимізації, оскільки вона досить добре працює в умовах дисбалансу класів. Головним підходом до побудови інформативних векторних представлень текстів було обрано мовну модель BERT для української мови. В ході процесу навчання ваги даної моделі залишались незмінними, а навчався лише останній шар моделі. Точність розробленої моделі складає 96.9%.

У розділі програмного та технічного забезпечення було описано основні засоби розробки, використані модулі, вимоги до технічного забезпечення та розроблену архітектуру програмного забезпечення. Розроблене програмне забезпечення потребує доволі суттєвих ресурсів для розгортання та коректної роботи. Архітектура програмного забезпечення представлена описом розроблених класів, діаграмою залежності розроблених пакетів, діаграмою розгортання та специфікацією функцій.

У технічному розділі було наведено керівництво користувача, де описано всі можливі дії, які може виконати користувач при користуванні розробленим сервісом. У підрозділі випробування програмного продукту було описано мету, загальні положення, наведено результати випробувань. Метою випробувань була перевірка коректності виконання процесу розпізнавання іменованих сутностей в текстах. Дані випробування проводились шляхом функціонального тестування. Було описано основні функції системи, перевірено їх відповідність вимогам технічного завдання. Для кожної з функціональних вимог прописано окреме випробування. У результаті випробувань було виправлено всі виявлені помилки.

Розроблений програмний продукт може бути корисним при роботі з неструктурованими текстовими даними та при наявності досить великої вибірки розмічених текстів, він може бути модифікований для роботи в певній доменній області. Також, даний сервіс може застосовуватись як мета-модель, тобто для покращення точності роботи інших, більш складних моделей.

Даний проєкт може бути розвинений більшим функціоналом для обробки текстів. А інтерфейс для роботи з даним сервісом може бути

додатково представлений як у модульному вигляді, так і у вигляді REST API. Також, при додатковій розмітці більшого набору текстів, є можливість покращити точність роботи моделі чи додати нові класи іменованих сутностей.

ПЕРЕЛІК ПОСИЛАНЬ

1. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Електронний ресурс] / J.Devlin, M. Chang, K. Lee, K. Toutanova. – 2018. – Режим доступу до ресурсу: https://arxiv.org/abs/1810.04805.
2. What is Named Entity Recognition [Електронний ресурс] – Режим доступу до ресурсу: https://www.techslang.com/definition/what-is-named-entity-recognition-ner/.
3. СТРУКТУРИРОВАННЫЕ И НЕСТРУКТУРИРОВАННЫЕ ДАННЫЕ: СРАВНЕНИЕ И ОБЪЯСНЕНИЕ [Електронний ресурс] – Режим доступу до ресурсу: https://asu-analitika.ru/strukturirovannye-i-nestrukturirovannye-dannye-sravnenie-i-objasnenie/.
4. spaCy 101: Everything you need to know [Електронний ресурс] – Режим доступу до ресурсу: https://spacy.io/usage/spacy-101.
5. NLTK book [Електронний ресурс] – Режим доступу до ресурсу: https://www.nltk.org/book/.
6. Проект Natasha. Набор качественных открытых инструментов для обработки естественного русского языка [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/516098/.
7. DeepPavlov для разработчиков: #1 инструменты NLP и создания чат-ботов [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/mipt/blog/472890/.
8. ner-ua [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/nazarii-piontko/ner-ua.
9. WordPiece [Електронний ресурс] – Режим доступу до ресурсу: https://paperswithcode.com/method/wordpiece.
10. Устройство CPython. Доклад Яндекса [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/yandex/blog/511972/.

11. What is Python? Executive Summary [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/doc/essays/blurb/.
12. McCormick C. BERT Fine-Tuning Tutorial with PyTorch [Електронний ресурс] / Chris McCormick. – 2019. – Режим доступу до ресурсу: https://mccormickml.com/2019/07/22/BERT-fine-tuning/.
13. What Is SQLite? [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlite.org/index.html.
14. What is Focal Loss and when should you use it? [Електронний ресурс] – Режим доступу до ресурсу: https://amaarora.github.io/2020/06/29/FocalLoss.html.
15. A Gentle Introduction to Cross-Entropy for Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/cross-entropy-for-machine-learning/.
16. Нейронные сети, перцептрон [Електронний ресурс] – Режим доступу до ресурсу: https://neerc.ifmo.ru/wiki/index.php?title=%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%81%D0%B5%D1%82%D0%B8,_%D0%BF%D0%B5%D1%80%D1%86%D0%B5%D0%BF%D1%82%D1%80%D0%BE%D0%BD.
17. TF-IDF с примерами кода: просто и понятно [Електронний ресурс] – Режим доступу до ресурсу: http://nlpx.net/archives/57.
18. Word2Vec: как работать с векторными представлениями слов [Електронний ресурс] – Режим доступу до ресурсу: https://neurohive.io/ru/osnovy-data-science/word2vec-vektornye-predstavlenija-slov-dlja-mashinnogo-obuchenija/.
19. LSTM — нейронная сеть с долгой краткосрочной памятью [Електронний ресурс] – Режим доступу до ресурсу: https://neurohive.io/ru/osnovy-data-science/lstm-nejronnaja-set/.

20. Туториал по PyTorch: от установки до готовой нейронной сети [Електронний ресурс] – Режим доступу до ресурсу: https://neurohive.io/ru/tutorial/glubokoe-obuchenie-s-pytorch/.
21. Hugging Face Transformers Package – What Is It and How To Use It [Електронний ресурс] – Режим доступу до ресурсу: https://www.kdnuggets.com/2021/02/hugging-face-transformer-basics.html.
22. ukr-roberta-base [Електронний ресурс] – Режим доступу до ресурсу: https://huggingface.co/youscan/ukr-roberta-base.
23. Optimization [Електронний ресурс] – Режим доступу до ресурсу: https://huggingface.co/transformers/main_classes/optimizer_schedules.html.
24. Learning Rate Schedules (Pytorch) [Електронний ресурс] – Режим доступу до ресурсу: https://huggingface.co/transformers/main_classes/optimizer_schedules.html#learning-rate-schedules-pytorch.
25. catalyst [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/catalyst-team/catalys
26. Poetry: новый менеджер зависимостей в Python [Електронний ресурс] – Режим доступу до ресурсу: https://khashtamov.com/ru/python-poetry-dependency-management/
27. Coding Neural Network — Forward Propagation and Backpropagtion [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/coding-neural-network-forward-propagation-and-backpropagtion-ccf8cf369f76
28. Understanding LSTM Networks [Електронний ресурс] – Режим доступу до ресурсу: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Додаток А

Тексти програмного коду

Сервіс з розпізнавання іменованих сутностей в текстах

(Найменування програми (документа))

DVD-R

(Вид носія даних)

30 арк, 244 Кб

(Обсяг програми (документа) , арк., Кб)

Київ – 2021 року

train.py

```
import json
from pathlib import Path

import joblib
import torch
from catalyst.utils import prepare_cudnn, set_global_seed
from sklearn import model_selection
from transformers import AdamW, get_linear_schedule_with_warmup

from datasets import NamedEntityRecognitionDataset
from models import NamedEntityRecognitionBertModel
from models.loss_fn import FocalLossCustom
from utils.callbacks import AccuracyCallbackCustom
from utils.helpers import get_config, remove_dir
from utils.runners import CustomRunner



PATH2ROOT = Path('..')
PATH2CONFIG = Path(PATH2ROOT / 'configs')
CONFIG = get_config(PATH2CONFIG / 'config.yml')
PATH2CORPUS = Path(PATH2ROOT /
CONFIG['data']['path_to_corpus_folder'])
MODEL_NAME = CONFIG['model']['model_name']



if __name__ == "__main__":
    data = joblib.load(PATH2ROOT / CONFIG['data']['path_to_preproc_data'])
    texts = data['text'].values.tolist()
    tags = data['tags'].values.tolist()
```

```
with open(PATH2CONFIG / 'target_mapper.json', 'r') as file:
    tag_map = json.load(file)

tag_map = {key: val[0] for key, val in tag_map.items()}

for i in range(len(tags)):
    for j in range(len(tags[i])):
        tags[i][j] = tag_map[tags[i][j]]

num_tag = len(tag_map.keys())

train_texts, val_texts, train_tags, val_tags = model_selection.train_test_split(
    texts,
    tags,
    random_state=CONFIG['general']['seed'],
    test_size=CONFIG['general']['test_size'],
)

train_dataset = NamedEntityRecognitionDataset(
    texts=train_texts,
    tags=train_tags,
    tokenizer=MODEL_NAME,
    max_seq_len=CONFIG['model']['max_seq_length'],
    lazy_mode=True,
)

val_dataset = NamedEntityRecognitionDataset(
    texts=val_texts,
    tags=val_tags,
```

```
tokenizer=MODEL_NAME,
            max_seq_len=CONFIG['model']['max_seq_length'],
            lazy_mode=True,
        )

        train_data_loader = torch.utils.data.DataLoader(
            train_dataset,
            batch_size=CONFIG['training']['train_batch_size'],
            num_workers=CONFIG['training']['num_workers'],
            shuffle=True,
        )

        val_data_loader = torch.utils.data.DataLoader(
            val_dataset,
            batch_size=CONFIG['training']['valid_batch_size'],
            num_workers=CONFIG['training']['num_workers'],
        )

        device = (
            torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
            if CONFIG['general']['device'] == 'auto'
            else torch.device(CONFIG['general']['device'])
        )

        model = NamedEntityRecognitionBertModel(
            pretrained_model_name=MODEL_NAME,
            output_dim=num_tag,
            lstm_dim=CONFIG['model']['lstm_dim'],
            lstm_num_layers=CONFIG['model']['lstm_num_layers'],
```

```
lstm_dropout_rate=CONFIG['model']['lstm_dropout_rate'],

lstm_bidirectional_flag=bool(CONFIG['model']['lstm_bidirectional_flag']),
            cnn_dropout_rate=CONFIG['model']['cnn_dropout_rate'],
            fc_dropout_rate=CONFIG['model']['fc_dropout_rate'],
            use_lstm_flag=CONFIG['model']['use_lstm_flag'],
            use_cnn_flag=bool(CONFIG['model']['use_cnn_flag']),
        )
        model = model.to(device)

        model.freeze()
        param_optimizer = list(filter(lambda p: p[1].requires_grad,
model.named_parameters()))
        model.unfreeze()

        no_decay = [
            "bias",
            "LayerNorm.bias",
            "LayerNorm.weight",
            'gamma',
            'beta',
            'final_layer_norm.weight',
        ]
        param_optimizer = [
            {
                "params": [
                    p for n, p in param_optimizer if not any(nd in n for nd in no_decay)
                ],
                "weight_decay": 0.001,
```

```
            },
            {
                "params": [p for n, p in param_optimizer if any(nd in n for nd in
no_decay)],
                "weight_decay": 0.0,
            },
        ]

        num_train_steps = int(
            len(train_texts)
            / CONFIG['training']['train_batch_size']
            * CONFIG['training']['num_epochs']
        )
        optimizer = AdamW(param_optimizer,
lr=float(CONFIG['training']['learning_rate']))

        scheduler = get_linear_schedule_with_warmup(
            optimizer,
            num_warmup_steps=CONFIG['training']['train_batch_size'] * 2,
            num_training_steps=num_train_steps,
        )

        if bool(CONFIG['training']['is_deterministic']):
            set_global_seed(CONFIG["general"]["seed"])
            prepare_cudnn(deterministic=True)

        remove_dir(PATH2ROOT / CONFIG["training"]["log_dir"])
        (PATH2ROOT / CONFIG["training"]["log_dir"]).mkdir()
        (PATH2ROOT / CONFIG["training"]["log_dir"] / '.gitkeep').touch()
```

```
        loaders = {"train":
train_data_loader, "valid": val_data_loader}

        model.freeze()

        runner = CustomRunner(
            custom_metrics={
                'accuracy': AccuracyCallbackCustom(),
            },
        )

        runner.train(
            model=model,
            optimizer=optimizer,
            scheduler=scheduler,

criterion=FocalLossCustom(gamma=CONFIG['training']['focal_loss_gamma']),
            loaders=loaders,
            num_epochs=CONFIG['training']['num_epochs'],
            logdir=PATH2ROOT / CONFIG["training"]["log_dir"],
            load_best_on_end=True,
            verbose=True,
            timeit=False,  # you can pass True to measure execution time of different
parts of train process
        )
        model.unfreeze()

        torch.save(
            model.state_dict(), PATH2ROOT / CONFIG['data']['path_to_logdir'] /
'best.pth'
```

```
        )
```

app.py

```
import streamlit as st

from pages import about_page, general_page, user_history_page
from pages.db import db_creation



if __name__ == '__main__':
    db_creation()

    options = {
        'General': general_page,
        'User History': user_history_page,
        'About': about_page,
    }

    st.sidebar.header("Please, choose page")
    page = st.sidebar.selectbox('', key='page_choice_box', options=list(options.keys()))

    options[page]()
```

NamedEntityRecognitionDataset.py

```
from typing import Dict, Iterable, Union

import torch
import transformers
from torch.utils.data import Dataset
from tqdm.auto import tqdm
from transformers import AutoTokenizer
```

```
class NamedEntityRecognitionDataset(Dataset):
    """
    Dataset for NER task.
    """

    def __init__(
        self,
        texts: Iterable[Iterable[str]],
        tags: Iterable[Iterable[str]] = None,
        tokenizer: Union[
            str, transformers.tokenization_utils.PreTrainedTokenizer
        ] = 'distilbert-base-uncased',
        max_seq_len: int = None,
        lazy_mode: bool = True,
    ):
        self.tags = tags
        self.texts = texts

        if isinstance(tokenizer, str):
            self.tokenizer = AutoTokenizer.from_pretrained(tokenizer)
        elif isinstance(tokenizer, transformers.tokenization_utils.PreTrainedTokenizer):
            self.tokenizer = tokenizer
        else:
            raise TypeError(
                "You pass wrong type of tokenizer. It should be a model name or PreTrainedTokenizer."
            )

        self.max_seq_len = max_seq_len
```

```
    self.length = len(texts)

    if self.max_seq_len < 3:
        raise ValueError("Max sequence length should be greather than 2")

    if not lazy_mode:
        self.encoded = [
            self._getitem_lazy(idx)
            for idx in tqdm(range(self.length), desc='tokenizing texts')
        ]
        del self.texts
        del self.tags

    self._getitem_fn = self._getitem_lazy if lazy_mode else self._getitem_encoded

def __len__(self) -> int:
    return self.length

def _getitem_encoded(self, index: int) -> Dict[str, torch.Tensor]:
    return self.encoded[index]

def _getitem_lazy(self, index: int) -> Dict[str, torch.Tensor]:
    sentence = self.texts[index]
    tag = self.tags[index]

    input_ids = []
    target_tag = []

    for i, word in enumerate(sentence):
        words_piece_ids = self.tokenizer.encode(
```

```
            word,
            max_length=self.max_seq_len,
            truncation=True,
            add_special_tokens=False,
        )
        input_ids.extend(words_piece_ids)
        target_tag.extend([tag[i]] * len(words_piece_ids))

    input_ids = (
        [self.tokenizer.cls_token_id]
        + input_ids[: self.max_seq_len - 2]
        + [self.tokenizer.sep_token_id]
    )

    attention_mask = [1] * len(input_ids)

    padding_len = self.max_seq_len - len(input_ids)
    input_ids = input_ids + ([self.tokenizer.pad_token_id] * padding_len)
    attention_mask = attention_mask + ([0] * padding_len)

    target_tag = [0] + target_tag[: self.max_seq_len - 2] + [0]
    target_tag = target_tag + ([0] * padding_len)

    return {
        'input_ids': torch.tensor(input_ids, dtype=torch.long),
        'attention_mask': torch.tensor(attention_mask, dtype=torch.long),
        'target_tag': torch.tensor(target_tag, dtype=torch.long),
    }
```

```
def __getitem__(self, index: int) -> Dict[str, torch.Tensor]:
```

```
        return self._getitem_fn(index)
```

NamedEntityRecognitionModel.py

```
from typing import Dict

import torch
import torch.nn as nn
from transformers import AutoConfig, AutoModel



class NamedEntityRecognitionBertModel(nn.Module):
    def __init__(
        self,
        pretrained_model_name: str,
        output_dim: int,
        lstm_dim: int = 256,
        lstm_num_layers: int = 2,
        lstm_dropout_rate: float = 0.3,
        lstm_bidirectional_flag: bool = True,
        cnn_dropout_rate: float = 0.4,
        fc_dropout_rate: float = 0.4,
        use_lstm_flag: bool = False,
        use_cnn_flag: bool = False,
    ):
        super().__init__()

        self.output_dim = output_dim
        self.use_lstm_flag = use_lstm_flag
        self.use_cnn_flag = use_cnn_flag

        config = AutoConfig.from_pretrained(pretrained_model_name)
```

```
        self.model = AutoModel.from_pretrained(pretrained_model_name,
config=config)

        if self.use_lstm_flag:
            self.lstm = nn.LSTM(
                self.model.config.hidden_size,
                lstm_dim,
                num_layers=lstm_num_layers,
                bidirectional=lstm_bidirectional_flag,
                batch_first=True,
                dropout=lstm_dropout_rate if lstm_num_layers > 1 else 0,
            )

        self.dropout = nn.Dropout(fc_dropout_rate)
        lstm_output_dim = lstm_dim * 2 if lstm_bidirectional_flag else lstm_dim
        lstm_output_dim = (
            lstm_output_dim * 2 if self.use_lstm_flag else self.model.config.hidden_size
        )

        self.fc = nn.Linear(lstm_output_dim, output_dim)

        if self.use_cnn_flag:
            self.cnn_list = list()
            for _ in range(1):
                self.cnn_list.append(
                    nn.Conv1d(
                        in_channels=lstm_output_dim,
                        out_channels=lstm_output_dim,
                        kernel_size=3,
```

```
                padding=1,
            )
        )
        self.cnn_list.append(nn.ReLU())
        self.cnn_list.append(nn.Dropout(cnn_dropout_rate))
        self.cnn_list.append(nn.BatchNorm1d(lstm_output_dim))
    self.cnn = nn.Sequential(*self.cnn_list)

def forward(
    self,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    **kwargs: Dict,
):
    logits, _ = self.model(
        input_ids, attention_mask=attention_mask, return_dict=False
    )

    # print('o1', o1.size()) [32, 128, 768] == [batch size, sent len, emb dim]
    # o1 = o1.permute(1, 0, 2) # [sent len, batch size, emb dim]

    if self.use_lstm_flag:
        logits, (hidden, cell) = self.lstm(logits)
    if self.use_cnn_flag:
        logits = (
            self.cnn(logits.transpose(2, 1).contiguous()).transpose(2, 1).contiguous()
        )
        logits = logits.permute(1, 0, 2)
    logits = self.fc(logits)
    return logits
```

```
    def freeze(self):
        for param in self.model.parameters():
            param.requires_grad = False

    def unfreeze(self):
        for param in self.model.parameters():
            param.requires_grad = True
```

CrossEntropyLossCustom.py

```
import torch.nn as nn
import torch.nn.functional as F


class CrossEntropyLossCustom(nn.modules.loss._WeightedLoss):
    def __init__(self, weight=None, reduction='mean'):
        super().__init__(weight, reduction=reduction)
        self.weight = weight

    def forward(self, output, target, attention_mask, output_dim):
        active_loss = attention_mask.view(-1) == 1
        active_logits = output.view(-1, output_dim)[active_loss]
        active_labels = target.view(-1)[active_loss]

        ce_loss = F.cross_entropy(
            active_logits, active_labels, reduction=self.reduction, weight=self.weight
        )
        return ce_loss
```

FocalLossCustom.py

```
import torch
import torch.nn as nn
```

```
import torch.nn.functional as F


class FocalLossCustom(nn.modules.loss._WeightedLoss):
    def __init__(self, weight=None, gamma=2, reduction='mean'):
        super().__init__(weight, reduction=reduction)
        self.gamma = gamma
        self.weight = weight

    def forward(self, output, target, attention_mask, output_dim):
        active_loss = attention_mask.view(-1) == 1
        active_logits = output.view(-1, output_dim)[active_loss]
        active_labels = target.view(-1)[active_loss]

        ce_loss = F.cross_entropy(
            active_logits, active_labels, reduction=self.reduction, weight=self.weight
        )
        pt = torch.exp(-ce_loss)
        focal_loss = ((1 - pt) ** self.gamma * ce_loss).mean()
        return focal_loss
```

db.py

```
import datetime as dt
import sqlite3 as sql
from typing import Tuple


def db_creation():
    conn = sql.connect('pages/db/user_history.db')
    cur = conn.cursor()
```

```
    cur.executescript(
        '''
        CREATE TABLE IF NOT EXISTS UserHistory(
            id INTEGER PRIMARY KEY,
            raw_text TEXT NOT NULL,
            tok_text TEXT NOT NULL,
            parameters TEXT NOT NULL,
            result TEXT NOT NULL,
            ins_dt timestamp NOT NULL
        );'''
    )

    conn.commit()
    conn.close()


def db_clean():
    conn = sql.connect('pages/db/user_history.db')
    cur = conn.cursor()

    cur.executescript('''DELETE FROM UserHistory;''')

    conn.commit()
    conn.close()


def db_insert(raw_text: str, tok_text: str, parameters: str, result: str):
    conn = sql.connect('pages/db/user_history.db')
    cur = conn.cursor()
```

```
    user_history = (raw_text,
tok_text, parameters, result, dt.datetime.now(tz=None))

    script = '''
        INSERT INTO UserHistory (raw_text, tok_text, parameters, result, ins_dt)
        VALUES(?, ?, ?, ?, ?);
    '''
    cur.execute(script, user_history)

    conn.commit()
    conn.close()



def db_select() -> Tuple:
    conn = sql.connect('pages/db/user_history.db')
    cur = conn.cursor()

    cur.execute(
        '''SELECT uh.raw_text, uh.tok_text, uh.parameters, uh.result, uh.ins_dt FROM
UserHistory as uh;'''
    )
    rows = cur.fetchall()

    conn.commit()
    conn.close()

    return rows
```

about_page.py

```
from pathlib import Path
```

```
import streamlit as st

from utils.helpers import get_config


def get_about_page():
    st.title('About')

    PATH2ROOT = Path('..')
    PATH2CONFIG = Path(PATH2ROOT / 'configs')
    CONFIG = get_config(PATH2CONFIG / 'config.yml')

    st.header('Model parameters')
    st.json(CONFIG)

    st.header('Loss function plot')
    st.markdown(
        '![](https://github.com/kryvokhyzha/bert-for-ukranian-
ner/blob/main/imgs/tb_focal_loss.png?raw=true)'
    )

    st.header('Accuracy plot')
    st.markdown(
        '![](https://github.com/kryvokhyzha/bert-for-ukranian-
ner/blob/main/imgs/tb_accuracy.png?raw=true)'
    )
```

general_page.py

```
import json
from pathlib import Path
from typing import Dict
```

```
import numpy as np
import streamlit as st
import torch

from datasets import NamedEntityRecognitionDataset
from models import NamedEntityRecognitionBertModel
from pages.db import db_insert
from utils.annotation import annotated_text
from utils.constants import TEXT_EXAMPLES
from utils.helpers import get_config


@st.cache(allow_output_mutation=True)
def get_model(PATH2ROOT: Path, CONFIG: Dict, device, output_dim):
    model = NamedEntityRecognitionBertModel(
        pretrained_model_name=CONFIG['model']['model_name'],
        output_dim=output_dim,
        lstm_dim=CONFIG['model']['lstm_dim'],
        lstm_num_layers=CONFIG['model']['lstm_num_layers'],
        lstm_dropout_rate=CONFIG['model']['lstm_dropout_rate'],
        lstm_bidirectional_flag=bool(CONFIG['model']['lstm_bidirectional_flag']),
        cnn_dropout_rate=CONFIG['model']['cnn_dropout_rate'],
        fc_dropout_rate=CONFIG['model']['fc_dropout_rate'],
        use_lstm_flag=bool(CONFIG['model']['use_lstm_flag']),
        use_cnn_flag=bool(CONFIG['model']['use_cnn_flag']),
    )

    model.load_state_dict(
        torch.load(
```

```
        PATH2ROOT                /
CONFIG['data']['path_to_logdir'] / 'best.pth', map_location=device
      )
  )

  model.freeze()
  return model


def get_tag_map(path: Path) -> Dict:
  with open(path / 'target_mapper.json', 'r') as file:
    tag_map = json.load(file)
  return tag_map


@st.cache
def get_inv_tag_map(tag_map: Dict) -> Dict:
  return {val[0]: (key, val[1]) for key, val in tag_map.items()}


@st.cache
def get_device(CONFIG: Dict):
  return (
    torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    if CONFIG['general']['device'] == 'auto'
    else torch.device(CONFIG['general']['device'])
  )


def get_prediction(model, test_dataset, device):
```

```
    model.eval()

    with torch.no_grad():
        for d in test_dataset:
            for k, v in d.items():
                d[k] = v.unsqueeze(0).to(device)

            outputs = model(**d)

    ann_text = list(
        zip(
            [
                test_dataset.tokenizer.decode(t)
                for t in d['input_ids'][0][1 : d['attention_mask'].view(-1).sum() - 1]
            ],
            outputs.argmax(2)
            .cpu()
            .numpy()
            .reshape(-1)[1 : d['attention_mask'].view(-1).sum() - 1],
        )
    )

    return ann_text


def get_general_page():
    PATH2ROOT = Path('..')
    PATH2CONFIG = Path(PATH2ROOT / 'configs')
    CONFIG = get_config(PATH2CONFIG / 'config.yml')
```

```
    tag_map                    =
get_tag_map(PATH2CONFIG)
    inv_tag_map = get_inv_tag_map(tag_map)

    device = get_device(CONFIG)
    model = get_model(PATH2ROOT, CONFIG, device, len(tag_map.keys()))

    st.title('Named Entity Recognition using BERT')
    st.subheader(
        'This simple application provide a way to visualize token classification task
(NER)💡'
    )

    text_key = st.selectbox(
        'select some example or type your text', list(TEXT_EXAMPLES.keys())
    )

    raw_text = st.text_area("enter text here", value=TEXT_EXAMPLES[text_key])
    raw_text_split = raw_text.split()

    test_dataset = NamedEntityRecognitionDataset(
        texts=[raw_text_split],
        tags=[[0] * len(raw_text_split)],
        tokenizer=CONFIG['model']['model_name'],
        max_seq_len=CONFIG['model']['max_seq_length'],
        lazy_mode=False,
    )

    if st.button('Submit🔥'):
```

```
    if raw_text == '':
        st.warning('Please, enter some text...')
        return
    elif len(raw_text_split) > CONFIG['model']['max_seq_length']:
        st.warning(
            f"Text should be shorter than {CONFIG['model']['max_seq_length']}
tokens..."
        )
        return
    ann_text = get_prediction(model, test_dataset, device)

    annotated_text(
        *[
            text + " "
            if class_ == tag_map['O'][0]
            else (text + " ", inv_tag_map[class_][0], inv_tag_map[class_][1])
            for text, class_ in ann_text
        ]
    )

    ann_text = np.array(ann_text)
    db_insert(
        raw_text,
        json.dumps(ann_text[:, 0].tolist(), ensure_ascii=False),
        json.dumps(CONFIG),
        json.dumps(ann_text[:, 1].tolist(), ensure_ascii=False),
    )
```

user_history_page.py

```
import pandas as pd
import streamlit as st
```

```
from pages.db import db_clean, db_select



def get_user_history_page():
    st.title('User history')

    rows = db_select()

    col = ('raw_text', 'tok_text', 'parameters', 'result', 'ins_dt')

    df = pd.DataFrame(rows, columns=col)

    columns = st.multiselect(
        label='What information do you want to display?', options=col, default=list(col)
    )

    if columns and df.shape[0]:
        st.write(df[columns])
        if st.button("Delete history"):
            db_clean()
    else:
        st.warning('User history is empty')
```

AccuracyCallbackCustom.py

```
from torch import nn



class AccuracyCallbackCustom(nn.Module):
    def __init__(self):
        super().__init__()
```

```
    def forward(self, output, target, attention_mask):
        active_loss = attention_mask.view(-1) == 1
        output_tags = output.argmax(2).view(-1)[active_loss]
        active_labels = target.view(-1)[active_loss]

        correct = active_labels.eq(output_tags)
        return correct.sum() / correct.shape[0]
```

CustomRunner.py

```
from catalyst import dl


class CustomRunner(dl.Runner):
    def __init__(self, custom_metrics):
        super().__init__()
        self.custom_metrics = custom_metrics

    def handle_batch(self, batch):
        # Unpack the data. Its structure depends on your model and
        # on what you pass to `train()`.

        y_pred = self.model(**batch)  # Forward pass

        target_tag = batch['target_tag']
        attention_mask = batch['attention_mask']
        # Compute the loss value
        loss = self.criterion(
            y_pred,
            target_tag,
            attention_mask=attention_mask,
```

```
output_dim=self.model.output_dim,
        )

        metrics_result = {'loss': loss}
        for name, metric_fn in self.custom_metrics.items():
            metrics_result[name] = metric_fn(y_pred, target_tag, attention_mask)

        # Update metrics (includes the metric that tracks the loss)
        self.batch_metrics.update(metrics_result)

        if self.is_train_loader:
            # Compute gradients
            loss.backward()
            # Update weights
            # (the optimizer is stored in `self.state`)
            self.optimizer.step()
            self.scheduler.step()
            self.optimizer.zero_grad()
```

annotation.py

```
import streamlit.components.v1
from htbuilder import HtmlElement, div, span, styles
from htbuilder.units import em, px, rem


def annotation(body, label="", background="#ddd", color="#333", **style):
    """Build an HtmlElement span object with the given body and annotation label.
    The end result will look something like this:
        [body | label]
```

```
----------
body : string
    The string to put in the "body" part of the annotation.
label : string
    The string to put in the "label" part of the annotation.
background : string
    The color to use for the background "chip" containing this annotation.
color : string
    The color to use for the body and label text.
**style : dict
    Any CSS you want to use to customize the containing "chip".
Examples
--------
Produce a simple annotation with default colors:
>>> annotation("apple", "fruit")
Produce an annotation with custom colors:
>>> annotation("apple", "fruit", background="#FF0", color="black")
Produce an annotation with crazy CSS:
>>> annotation("apple", "fruit", background="#FF0", border="1px dashed red")
"""

if "font_family" not in style:
    style["font_family"] = "sans-serif"

return span(
    style=styles(
        background=background,
        border_radius=rem(0.33),
        color=color,
        padding=(rem(0.17), rem(0.67)),
```

```
            display="inline-flex",
            justify_content="center",
            align_items="center",
            **style,
        )
    )(
        body,
        span(
            style=styles(
                color=color,
                font_size=em(0.67),
                opacity=0.5,
                padding_left=rem(0.5),
                text_transform="uppercase",
                margin_bottom=px(-2),
            )
        )(label),
    )


def annotated_text(*args, **kwargs):
    """Writes test with annotations into your Streamlit app.
    Parameters
    ----------
    *args : str, tuple or htbuilder.HtmlElement
        Arguments can be:
        - strings, to draw the string as-is on the screen.
        - tuples of the form (main_text, annotation_text, background, color) where
          background and foreground colors are optional and should be an CSS-valid
string such as
```

```
        "#aabbcc"  or  "rgb(10,  20,
30)"
        - HtmlElement objects in case you want to customize the annotations further. In
particular,
        you can import the `annotation()` function from this module to easily produce
annotations
        whose CSS you can customize via keyword arguments.
    """
    out = div(
        style=styles(
            font_family="sans-serif",
            line_height="1.5",
            color="white",
            font_size=px(16),
        )
    )

    for arg in args:
        if isinstance(arg, str):
            out(arg)

        elif isinstance(arg, HtmlElement):
            out(arg)

        elif isinstance(arg, tuple):
            out(annotation(*arg))

        else:
            raise Exception("Oh noes!")
```

```
streamlit.components.v1.html(str(out), **kwargs)
```

constants.py

```
TEXT_EXAMPLES = {
    'Empty text area': '',
    'Example 1': 'Привіт, мене звати Володимир. Я працюю в ТОВ "БРСМ", що знаходиться в місті Житомир',
    'Example 2': 'Водночас сьогодні вони можуть отримати лише подяку, грамоту та почесну грамоту, «Знак пошани».',
    'Example 3': 'Пішли Іван Гиба , Семен Сорока й дядько Тимухтей , взяли Пудика за чемери – і як був він у сорочці і штанях , так до корчми і притягли городами , щоб жінка не побачила .',
}
```

helpers.py

```
import os
import shutil
from pathlib import Path
from typing import Dict

import yaml


def get_config(filename: Path) -> Dict:
    with open(filename, 'r') as file:
        config = yaml.load(file, Loader=yaml.FullLoader)
    return config


def remove_dir(path: Path):
    """ param <path> could either be relative or absolute. """
    if os.path.isfile(path) or os.path.islink(path):
```

```
        os.remove(path)  # remove the
file
    elif os.path.isdir(path):
        shutil.rmtree(path)  # remove dir and all contains
    else:
        raise ValueError("file {} is not a file or dir.".format(path))
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО
Керівник проєкту
_____ *Юрій ОЛІЙНИК*
(підпис) (вл. ім’я, прізвище)
“5” квітня 2021 р.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)
“6” квітня 2021 р.

Сервіс з розпізнавання іменованих сутностей в текстах

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 7217.01.000 ТЗ*

на 12 сторінках

Київ – 2021 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ 4

1.1 Повне найменування системи та її умовне позначення 4

1.2 Найменування організації-замовника та організацій-учасників робіт.. 4

1.3 Перелік документів, на підставі яких створюється система 4

1.4 Планові терміни початку і закінчення роботи зі створення системи 4

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ 5

2.1 Призначення засобів 5

2.2 Цілі створення засобів 5

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ 6

4 ВИМОГИ ДО СИСТЕМИ 7

4.1 Вимоги до системи в цілому 7

4.1.1 Вимоги до надійності 7

4.1.2 Вимоги до збереження інформації 7

4.2 Вимоги до функціональних характеристик 7

4.3 Вимоги до видів забезпечення 8

4.3.1 Вимоги до математичного забезпечення 8

4.3.2 Вимоги до інформаційного забезпечення 8

4.3.3 Вимоги до прикладного програмного забезпечення 8

4.3.4 Вимоги до системного програмного забезпечення 9

4.3.5 Вимоги до технічного забезпечення 9

5 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ 11

5.1 Об’єм та види випробування 11

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повне найменування системи: Сервіс з розпізнавання іменованих сутностей в текстах.

Коротке найменування системи: «NER UA BERT».

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовником системи є кафедра автоматизованих систем обробки інформації і управління факультету інформатики та обчислювальної техніки НТУУ «КПІ ім. Ігоря Сікорського». Розробником системи є Кривохижа Роман Андрійович.

1.3 Перелік документів, на підставі яких створюється система

Підставою для розробки “Сервісу з розпізнавання іменованих сутностей” є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над створенням системи – 1 грудня 2020 року.

Плановий термін по закінченню роботи над створенням системи – 25 травня 2021 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення засобів

Система призначена для розпізнавання іменованих сутностей в текстах для української мови.

2.2 Цілі створення засобів

Метою даної системи є покращення розпізнавання іменованих сутностей в текстах для української мови.

Для досягнення наведеної мети необхідно реалізувати такі задачі:

а. збір даних для формування навчальної та тестової вибірки:
   1) виконання обробки зібраних даних;
   2) перетворення оброблених даних в простір індексів.

б. розробка модуля з розпізнавання іменованих сутностей:
   1) розробка модуля для завантаження та оперування даними;
   2) розробка архітектури моделі розпізнавання іменованих сутностей;
   3) реалізація підходу fine-tuning для моделі BERT з навчанням лише останнього шару;
   4) реалізація збору метрик.

в. розробка інтерфейсу для взаємодії користувача з розробленою моделлю:
   1) розробити інтерфейс для введення тексту для подальшої класифікації токенів;
   2) розробити інтерфейс для відображення історії користування моделлю;
   3) розробити інтерфейс для відображення результатів навчання моделі.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Об'єктом автоматизації є процес розпізнавання іменованих сутностей в текстах для української мови, а також візуалізація отриманих результатів. Створення даної системи може значно полегшити роботу з неструктурованими текстовими даними, покращити роботу мета-моделей машинного навчання.

4 ВИМОГИ ДО СИСТЕМИ

4.1 Вимоги до системи в цілому

Сервіс має надавати користувачу можливість виконувати розпізнавання іменованих сутностей для української мови.

4.1.1 Вимоги до надійності

Для коректної роботи з системою технічні засоби користувачів мають мати стабільне підключення до мережі Інтернет та мати базові засоби введення та виведення інформації:

- монітор;
- клавіатура;
- комп’ютерна миша.

4.1.2 Вимоги до збереження інформації

Для запобігання втрати інформації при аварійних ситуацій мають створюватися репліки сховища даних, які можна потім використати для відновлення.

4.2 Вимоги до функціональних характеристик

Система має реалізовувати наступні функції:

а. зчитування введених у відповідне поле текстових даних;

б. виконання попередньої обробки даних:

1) виконання процесу розбиття тексту на слова;
2) виконання процесу розбиття слів на підслова;
3) переведення отриманих підслів в простір індексів.

в. виконання задачі класифікації токенів:

1) побудова інформативних векторних представлень для кожного токену;
2) отримання ймовірностей належності токена до певної іменованої сутності.

г. виконання процесу навчання моделі:

1) функція заморожування ваг мовної моделі;
2) функція навчання моделі.

4.3 Вимоги до видів забезпечення

4.3.1 Вимоги до математичного забезпечення

Система має містити можливість тренування ваг нейронної мережі та можливість їх застосування для вирішення задачі розпізнавання іменованих сутностей.

4.3.2 Вимоги до інформаційного забезпечення

Для тренування нейронної мережі мають бути зібрані набори текстів з розміченими іменованими сутностями. Результатом нейронної мережі є тензор logits-ймовірностей класів іменованих сутностей.

Для збереження результатів роботи моделі має бути використана реляційна СКБД загального призначення SQLite. Ця СКБД має використовуватись веб-додатком.

4.3.3 Вимоги до прикладного програмного забезпечення

Для коректної роботи з системою користувачі мають мати комп’ютер зі встановленим одним з браузерів, перелічених нижче, з версією, що вказана, або вище:

- Google Chrome 85;
- Microsoft Edge 88;
- Mozilla Firefox 86;

– Internet Explorer 11;
– Opera 73;
– Safari 8.

4.3.4 Вимоги до системного програмного забезпечення

Для розгортання архітектурний складових користувачу необхідно встановити Docker версії 19.0. Операційна система, що має використовуватись при розгортанні – Ubuntu 18.04.

4.3.5 Вимоги до технічного забезпечення

Технічні засоби, що використовуються під час проведення дослідження: віртуальна машина, що включає:

– процесор Intel Xeon 2.2Ггц або кращий;
– оперативна пам’ять – 8 гб або більше;
– жорсткий диск – 4 Гб вільного місця або більше.

Програмні засоби, що використовуються під час проведення випробувань:

– операційна система Pop OS версії 20.4 або вище;
– Python 3.8;
– пакетний менеджер Poetry;
– встановлено всі бібліотеки, що містяться в конфігураційному файлі відповідного пакетного менеджера.

СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з розробки наведені в таблиці 5.1.

Таблиця 5.1 - Основні етапи виконання робіт

№	Назва етапу	Термін виконання
1	Вивчення рекомендованої літератури	01.03.2021
2	Аналіз існуючих методів розв’язання задачі	07.03.2021

3	Постановка та формалізація задачі	15.03.2021
4	Розробка програмного забезпечення	11.05.2021
5	Налагодження програми	12.05.2021
6	Оформлення пояснювальної записки	13.05.2021
7	Подання ДП на попередній захист	14.05.2021
8	Подання ДП на основний захист	04.06.2021

5 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Приймання програмного продукту має відбутися 14.05.2021.

Подання готового дипломного проєкту на основний захист має відбутися 04.06.2021.

5.1 Об’єм та види випробування

Етапи випробувань:

- ознайомчий;
- випробування.

На ознайомчому етапі проводиться:

- перевірка комплектності програмної документації;
- перевірка комплектності складу технічних і програмних засобів.

Під час етапу випробувань проводиться:

- перевірка відповідності технічних характеристик системи;
- перевірка ступеню виконання вимог функціонального призначення сервісу.

Функції, що підлягають перевірці:

- функція зчитування вхідних даних та їх подальша обробка;
- функція виконання процесу навчання;
- функція класифікації нових текстів;
- функція відображення результату роботи моделі.

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008093271

Дата перевірки:
31.05.2021 02:35:23 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
31.05.2021 18:32:29 EEST

ID користувача:
77149

Назва документа: **Krivohija_bachelor_is72**

Кількість сторінок: **47** Кількість слів: **8854** Кількість символів: **66457** Розмір файлу: **659.35 KB** ID файлу: **1008178178**

7.85%
Схожість

Найбільша схожість: **1.06%** з джерелом з Бібліотеки (ID файлу: **1008167642**)

4.53% Джерела з Інтернету	138	Сторінка 49
7.75% Джерела з Бібліотеки	686	Сторінка 51

0.26% Цитат

Цитати	1	Сторінка 52

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

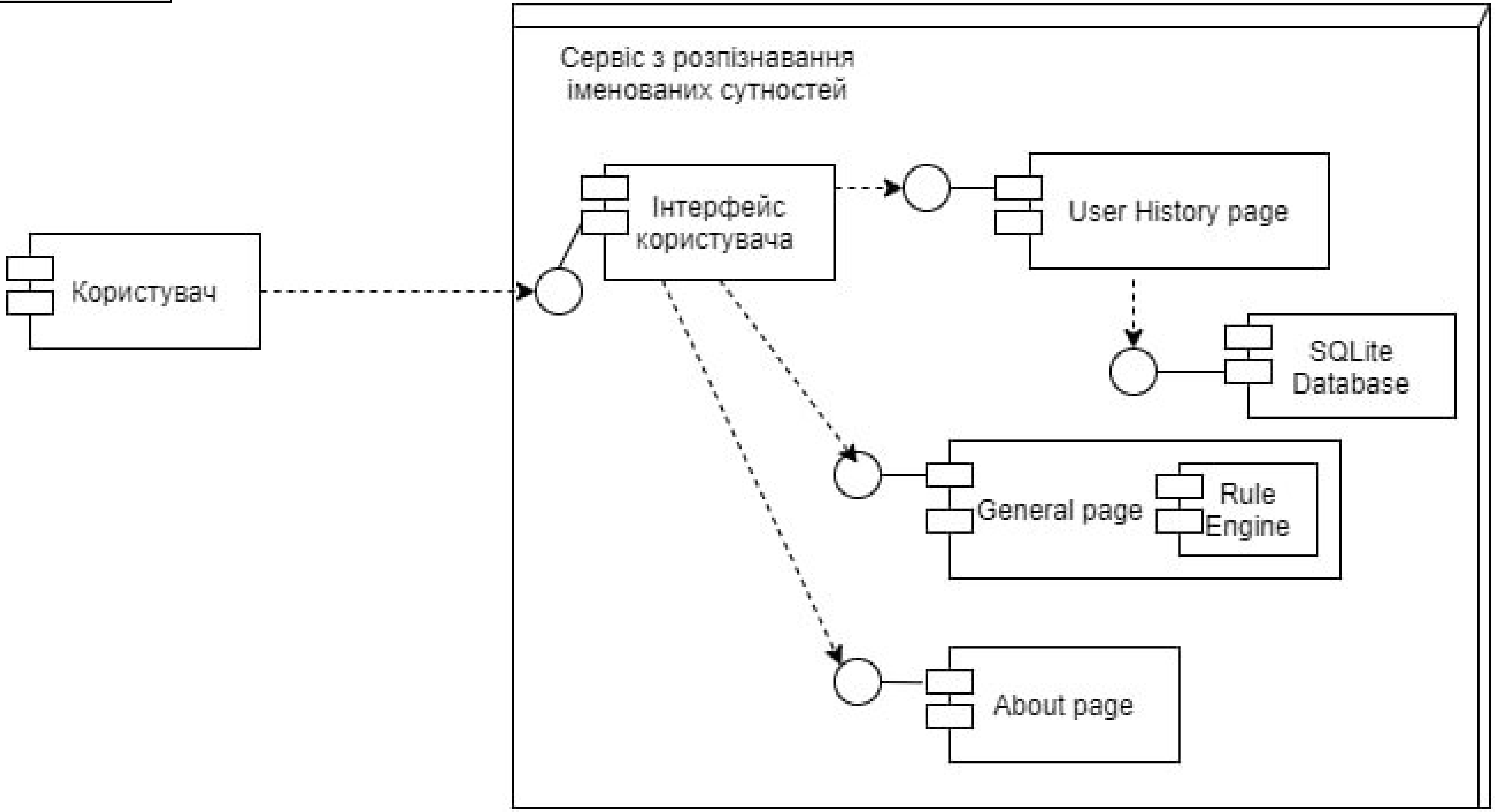
Замінені символи	28

Графічний матеріал
до дипломного проєкту

на тему: Сервіс з розпізнавання іменованих сутностей в текстах

Київ – 2021 року

ДП 7217.00.000 ПЗ



Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Кривохижа Р.А.		
Перевірив		Олійник Ю.О.		
Консульт.				
Н. кон.		Новінський В.П.		
Затвердив		Олійник Ю.О.		

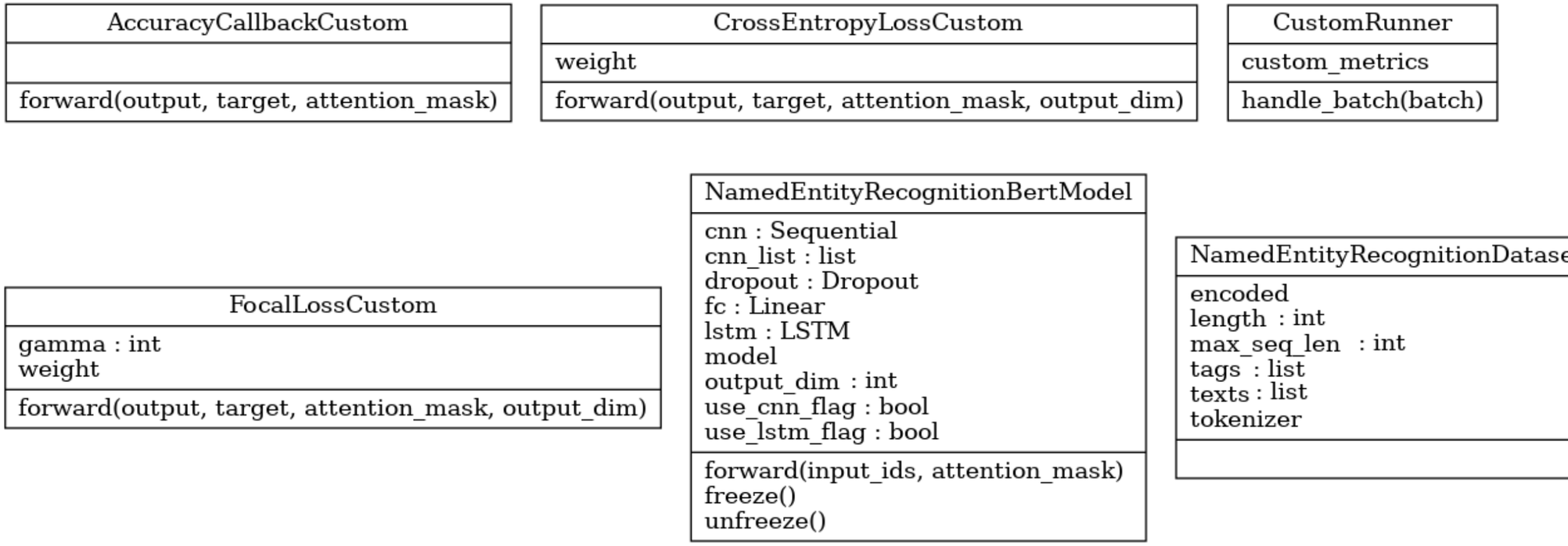
ДП 7217.06.000 ССК

Схема структурна компонентів

Літера	Маса	Масштаб
Аркуш 1	Аркушів 1	

Сервіс з розпізнавання іменованих сутностей в текстах

КПІ ім. Ігоря Сікорського
Каф. АСОІУ
Гр. ІС-72



					ДП 7217.03.000 ССК			
					Схема структурна класів програмного	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Кривохижа Р.А.				Аркуш 1	Аркушів 1	
Перевірив		Олійник Ю.О.						
Консульт.					Сервіс з розпізнавання іменованих сутностей в текстах	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-72		
Н. кон.		Новінський В.П.						
Затвердив		Олійник Ю.О.						

Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Кривохижа Р.А.		
Перевірив		Олійник Ю.О.		
Консульт.				
Н. кон.		Новіцький В.Л.		
Затвердив		Олійник Ю.О.		

ДП 7217.04.000 ССЗ

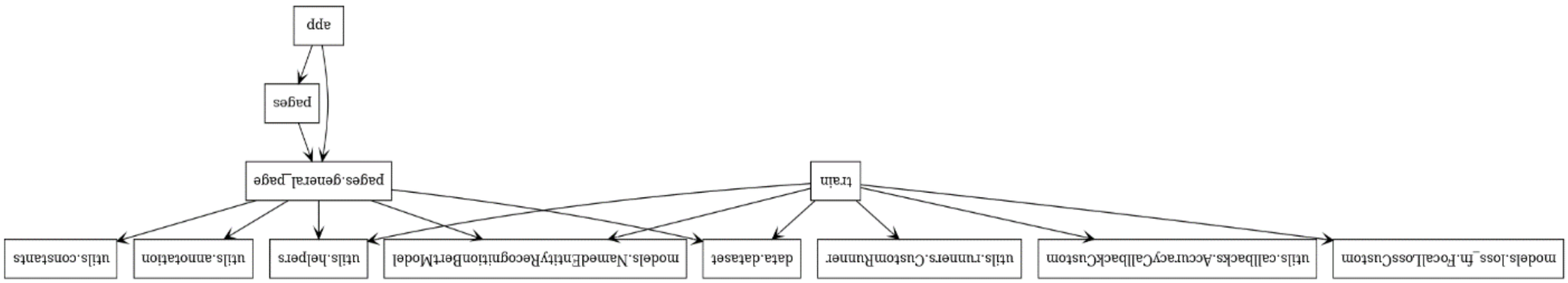
Літера	Маса	Масштаб

Схема структурна залежностi модулiв

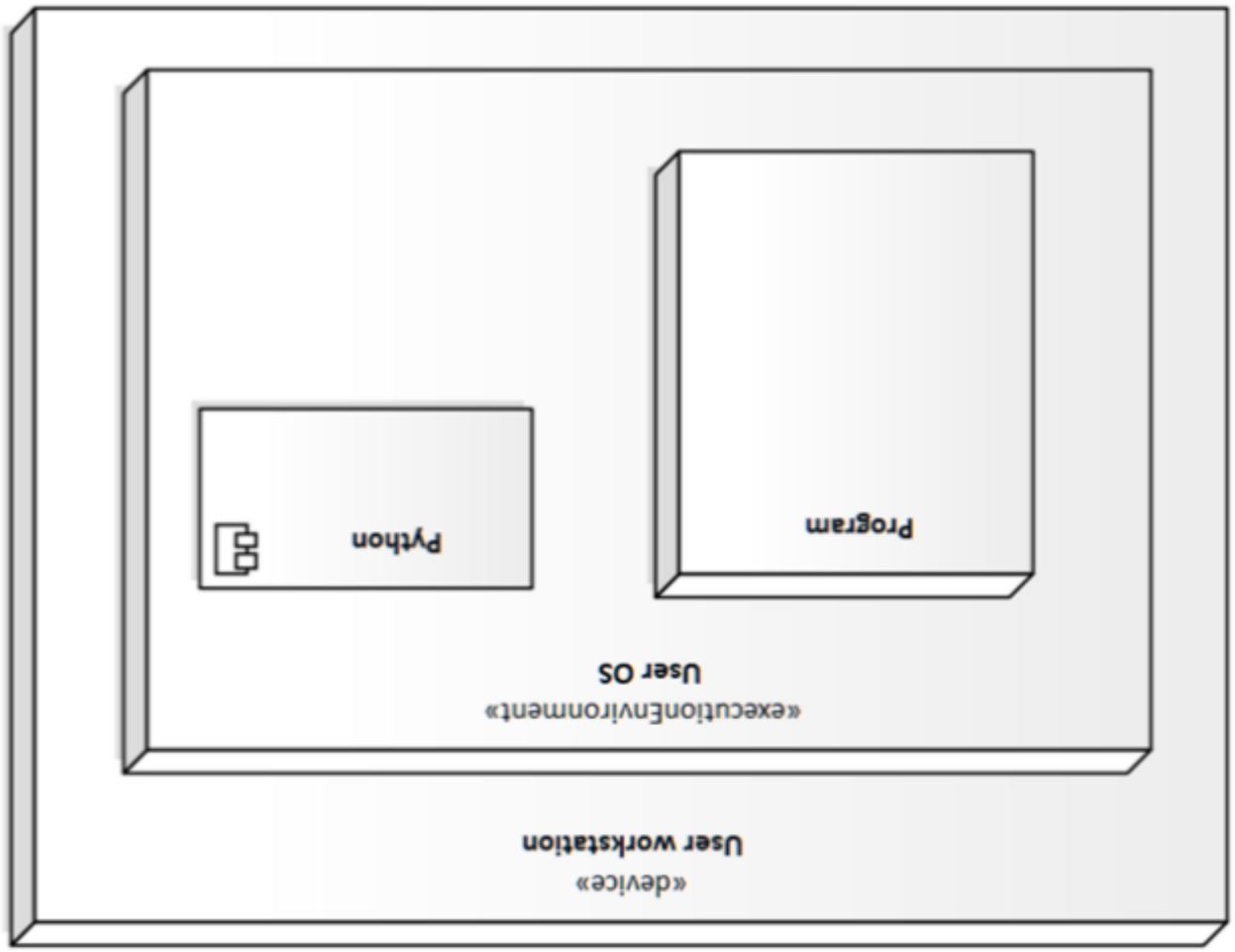
Аркуш 1	Аркушів 1

Сервіс з розпізнавання іменованих сутностей в текстах

КПІ ім. Ігоря Сікорського ФАС. АСОІУ Гр. ІС-72



ДП 7217.00.000 ПЗ



Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Кривохижа Р.А.			
Перевірив		Олійник Ю.О.			
Консульт.					
Н. кон.		Новіцький В.Л.			
Затвердив		Олійник Ю.О.			

ДП 7217.07.000 ССР

Літера	Маса	Масштаб

Схема структурна розгортання

Аркуш 1	Аркушів 1

Сервіс з розгінавання іменованих сутностей в мережах

КПІ ім. Ігоря Сікорського
Каф. АСОІУ
Гр. ІС-72

ДП 7217.00.000 ПЗ

					ДП 7217.02.000 ССД			
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера	Маса	Масштаб
Розробив		Кривохижа Р.А.						
Перевірив		Олійник Ю.О.				Аркуш 1	Аркушів 1	
Консульт.					Сервіс з розпізнавання іменованих сутностей в текстах	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-72		
Н. кон.		Новінський В.П.						
Затвердив		Олійник Ю.О.						