Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж»

Виконав: студент 2-го курсу, групи 1ПІ-20м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Позур М.Ю.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ:
Коваленко О.О.
(прізвище та ініціали)
«____» _____ 2021 р.

Опонент к.т.н., ст. викл. каф. КН:
Озеранський В. С.
(прізвище та ініціали)
«____» _____ 2021 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н, проф. Романюк О.Н.
(прізвище та ініціали)
«__» _____ 2021 р.

ВНТУ – 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 інформаційні технології
Спеціальність – 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«13» вересня 2021 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙУ РОБОТУ СТУДЕНТУ

Позуру Михайлу Юрійовичу

1. Тема роботи – розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж.

Керівник роботи: Коваленко Олена Олексіївна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від
«24» вересня 2021 року № 277.

2. Строк подання студентом роботи
1 грудня 2021 року

3. Вихідні дані до роботи: середовища розробки Visual Studio 2019 та Visual Studio Code, мови розробки C# та Python, операційна система – Windows 10, базові алгоритми для пошуку іменованих сутностей у тексті.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; аналіз моделей мови; розробка архітектури та алгоритмів системи; розробка методу підготовки вхідних даних; розробка методу пошуку іменованих сутностей; розробка програмного модуля для пошуку іменованих сутностей у

тексті, розробка веб-сервісу; тестування додатку; економічна частина; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу: блок-схеми алгоритмів роботи додатку; структура нейронної мережі; тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Коваленко О. О., к.т.н., доцент кафедри ПЗ		
5	Ратушняк О.Г., к.т.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання _____14 вересня 2021 р._____

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз архітектур нейронних мереж та вибір найбільш доцільної для поставленої задачі	15.09.2021 – 30.09.2021	Вик.
2	Розробка алгоритмів системи	01.10.2021 – 10.10.2021	Вик.
3	Розробка методу обробки вхідних даних	11.10.2021 – 25.10.2021	Вик.
4	Розробка методу пошуку іменованих сутностей	26.10.2021 – 15.11.2021	Вик.
5	Економічна частина	16.11.2021 – 30.11.2021	Вик.

Студент _____ Позур М. Ю.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Коваленко О. О.
(підпис) (прізвище та ініціали)

Анотація

УДК 004.912.032.26

Позур М.Ю. Розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2021. 134 с.

На укр. мові. Бібліогр.: 23 назв; рис.: 81; табл. 9.

У магістерській кваліфікаційній роботі проведено детальний аналіз методів розпізнавання сутностей у тексті.

Запропоновано використовувати нейронну мережу на основі BERT-BiLSTM-CRF, яка використовує модель мови CharacterBERT.

Розроблено метод пошуку іменованих сутностей, що використовує модель мови CharacterBERT та метод обробки вхідних даних, який розділяє вхідний текст на окремі речення, чим вдається покращити надійність роботи системи. Розроблено модуль для пошуку іменованих сутностей у тексті з використанням мови програмування Python, бібліотеки PyTorch та середовища розробки Visual Studio Code. Розроблено веб-сервіс, що використовує модуль розпізнавання іменованих сутностей. При розробці використано мову програмування C# та технології ASP.NET Core.

Отримані в магістерській кваліфікаційній роботі результати можна використати для побудови системи автоматичної обробки текстових даних.

Ключові слова: нейронні мережі, пошук іменованих сутностей, веб-технології, обробка природної мови.

Abstract

Pozur M.Y. Development of methods for named entity recognition web system using neural networks. Vinnitsa: VNTU, 2021. – 134 p.

In Ukrainian language. Bibliographer: 23 titles; fig.: 81; tabl. 9.

In the master's thesis, a detailed analysis of Named Entity Recognition methods was carried out.

A neural network based on BERT-BiLSTM-CRF that uses CharacterBERT language model was proposed as a solution.

Named entity recognition method that uses CharacterBERT language model and input data processing method that splits text into sentences were developed. Named Entity Recognition module was developed with usage of Python programming language, PyTorch library and Visual Studio Code IDE. Web service that uses Named Entity Recognition module was developed with usage of C# programming language and ASP.NET Core web framework.

Results obtained in the master's thesis can be used to build fully automated text data processing system.

Keywords: neural networks, named entity recognition, web services, natural language processing.

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ СТАНУ РОЗВИТКУ СИСТЕМ ПОШУКУ ІМЕНОВАНИХ СУТНОСТЕЙ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ 12
1.1 Аналіз стану пошуку іменованих сутностей у тексті 12
1.2 Порівняльний аналіз аналогів 13
1.3 Аналіз методів розв’язання поставленої задачі 15
1.4 Аналіз методів NER з використанням нейронних мереж 16
1.5 Постановка задачі для розпізнавання іменованих сутностей у тексті 21
1.6 Висновки 21
2 РОЗРОБКА МЕТОДІВ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ ПОШУКУ ІМЕНОВАНИХ СУТНОСТЕЙ 22
2.1 Аналіз моделей мови 22
2.2 Розробка алгоритмів роботи системи 26
2.3 Розробка методу підготовки вхідних даних 28
2.4 Розробка методу пошуку іменованих сутностей 32
2.5 Розробка алгоритму навчання нейронної мережі 37
2.6 Висновки 39
3 РОЗРОБКА СИСТЕМИ ПОШУКУ ІМЕНОВАНИХ СУТНОСТЕЙ У ТЕКСТІ 40
3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу. 40
3.2 Розробка нейронної мережі 42
3.3 Розробка модуля обробки даних 47
3.4 Розробка модуля навчання нейронної мережі 55
3.5 Розробка веб-сервісу 64
3.6 Висновки 70
4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ 71
4.1 Тестування системи 71
4.2 Розробка інструкції користувача 74

4.3 Висновки 76
5 ЕКОНОМІЧНА ЧАСТИНА 77
5.1 Оцінювання комерційного потенціалу розробки 77
5.2 Прогнозування витрат на виконання науково-дослідної роботи 82
5.3 Розрахунок економічної ефективності науково-технічної розробки 87
5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності 88
5.5 Висновки до економічного розділу 90
ВИСНОВКИ 92
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 93
ДОДАТКИ 96
Додаток А – Технічне завдання 97
Додаток Б – Протокол перевірки на плагіат 102
Додаток В – Лістинг програми 103
Додаток Г – Ілюстративна частина 126

ВСТУП

Обґрунтування вибору теми дослідження. Інформація є одним із найбільш цінних ресурсів сьогодення, адже її можна використати з метою оптимізувати певні процеси на підприємстві, покращити маркетинг компанії, спрогнозувати ціни на акції, тощо. Завдяки мережі інтернет процес збору даних став значно простішим, що дозволяє отримувати великі об’єми даних за лічені години. Через це проблема автоматизації обробки даних є досить гострою, адже потрібно створювати нові ефективні методи аналізу.

Інформація та дані, бувають різного виду, починаючи зі звичайних числових даних, закінчуючи відеорядом. У залежності від типу даних змінюються й методи їх обробки, наприклад, для обробки статистики буде достатньо звичайних математичних операцій, тоді як для обробки відеоряду необхідно використовувати більш складні алгоритми.

Одним із найпоширеніших видів інформації є текстова інформація. Її можна знайти будь-де, починаючи з електронних документів, закінчуючи коментарями в інтернеті. Задача обробки тексту належить до класу задач обробки природної мови (Natural Language Processing). NLP включає у себе великий спектр задач, що стосуються обробки природної мови, починаючи від звичайного класифікатору, закінчуючи системами, що здатні давати відповіді на запитання [1]. Задачі обробки природної мови потребують складних алгоритмів, так як необхідно розуміти, про що саме йдеться у тексті. Тому для таких задач, як правило, використовують нейронні мережі.

Однією із основних задач NLP є розпізнавання іменованих сутностей. Named Entity Recognition або NER – одна із задач видобутку інформації, що полягає у пошуку іменованих сутностей у неструктурованому тексті [2]. Категорії сутностей визначаються заздалегідь та залежать від даних, на яких навчається нейронна мережа. Ключовими є наступні категорії: імена людей, організації, локацій, назви подій та творів мистецтва. У залежності від конкретного випадку, список категорії може змінюватись. NER дозволяє

отримувати із тексту корисну інформацію за рахунок визначення ключових сутностей, які трапляються у тексті.

Як правило, системи для пошуку іменованих сутностей є частиною хмарних сервісів, що надають доступ до функціоналу з пошуку та обробки інформації. Головним недоліком таких сервісів є те, що вони використовують значно простіші алгоритми обробки та пошуку інформації, ефективність яких може бути покращена. Наприклад, у роботі «Named Entity Recognition Using BERT BiLSTM CRF for Chinese Electronic Health Records» [3] розглянуто архітектуру нейронної мережі для пошуку іменованих сутностей, що використовує модель мови BERT. Така архітектура є менш стійкою до помилок у тексті, ніж архітектура, що використовує модель мови CharacterBERT. Тому розробка власної системи є актуальною.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення надійності та реалістичності пошукового процесу іменованих сутностей шляхом використання нових методів обробки вхідних даних та методів пошуку іменованих сутностей, що використовують модель мови CharacterBERT.

Основними задачами роботи є:

- визначити найбільш ефективний підхід до пошуку іменованих сутностей у тексті;
- розробити метод підготовки вхідних даних;
- розробити метод пошуку іменованих сутностей у тексті;
- розробити модуль пошуку іменованих сутностей;
- розробити веб-сервіс для роботи з модулем пошуку іменованих сутностей;
- провести тестування програмного додатку.

Об'єктом дослідження є процес пошуку іменованих сутностей.

Предметом дослідження є методи та засоби пошуку іменованих сутностей у тексті.

Методи дослідження. У процесі досліджень використовувались методи дослідження:

- методи побудови та навчання нейронних мереж для побудови нейронної мережі;
- методи статистичного моделювання для визначення вихідного шару нейронної мережі;
- методи обробки та моделювання природної мови для побудови числової моделі тексту;
- методи регуляризації нейронних мереж для підвищення надійності роботи мережі.

Наукова новизна отриманих результатів.

- подальшого розвитку отримав метод розпізнавання іменованих сутностей, що базується на використанні BERT-BiLSTM-CRF нейронної мережі, у якому, на відміну від існуючих, використовується модель мови CharacterBERT, чим вдається підвищити надійність роботи системи;
- подальшого розвитку отримав метод підготовки даних до пошуку іменованих сутностей, у якому, на відміну від існуючих, текст розбивається на речення, що дозволяє дискретизувати вхідні дані й підвищити надійність пошуку іменованих сутностей.

Практична цінність отриманих результатів. Практичне значення магістерської кваліфікаційної роботи полягає у можливості практичного використання розроблених алгоритмів і веб-сервісу для пошуку іменованих сутностей у процесі обробки текстової інформації.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У науковій роботі, опублікованій у співавторстві, автору належать такі результати: модуль пошуку іменованих сутностей у тексті [4]; алгоритм автоматичної обробки

текстової інформації, що використовує модуль пошуку іменованих сутностей; метод пошуку іменованих сутностей у тексті, що використовує модель мови CharacterBERT [5].

Апробація матеріалів магістерської кваліфікаційної роботи. Результати магістерської кваліфікаційної роботи доповідалися та обговорювалися на:

- міжнародній науково-практичній конференції молодих вчених та студентів «Молодь у світі сучасних технологій» (Херсон 2020);
- міжнародній науково-практичній інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова» (Вінниця 2021).

Публікації. Основні результати дослідженні опубліковані в наступних наукових роботах

- тези доповіді на конференції «Молодь у світі сучасних технологій» [4];
- тези доповіді на інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова» [5].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку літератури, що містить 23 найменування, 4 додатки. Робота містить 81 ілюстрацію, 9 таблиць.

1 АНАЛІЗ СТАНУ РОЗВИТКУ СИСТЕМ ПОШУКУ ІМЕНОВАНИХ СУТНОСТЕЙ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану пошуку іменованих сутностей у тексті

Більшість компанії, які тим чи іншим чином працюють з текстовими даними, використовують системи обробки природної мови, до яких входять системи розпізнавання іменованих сутностей. У таких системах розпізнавання іменованих сутностей відіграє ключову роль, адже вона дозволяє встановити про що саме йдеться у тексті, що значно полегшує семантичний аналіз тексту та його розуміння в цілому.

Така інформація є досить цінною, особливо коли мова йде про аналіз великої кількості даних. Наприклад, використовуючи систему розпізнавання сутностей, можна проаналізувати відсоток статей, опублікованих у новинах за останній тиждень, у яких мова йшла про певну подію. Подібні системи досить зручно використовувати для аналізу тенденцій, що на сьогоднішній день є надзвичайно актуальним. Через значні переваги, які надають системи автоматичного аналізу природної мови, сьогодні ведуться активні дослідження у цій галузі, у тому числі й пошуку іменованих сутностей.

Не дивлячись на значний технологічний прогрес, та активне використання систем штучного інтелекту, питання пошуку іменованих сутностей у тексті не є остаточно вирішеним. Подібні системи, як правило, використовують алгоритми на основі нейронних мереж. Нейронна мережа потребує певного процесу навчання, для якого потрібні дані. Створення таких об’ємів даних також є проблемою, тому зараз ведеться активна робота над розробкою алгоритмів напівавтоматичного навчання нейронних мереж для розпізнавання іменованих сутностей. Частково із цього випливає проблема підтримання багатьох мов [6].

Хоча сьогодні й досі існують відкриті питання стосовно розробки систем розпізнавання іменованих сутностей, але вони не є критичними. Більшість тексту сьогодні публікується на англійській мові, що частково нівелює проблему з

підтримкою деяких мов. Деякі компанії вже використовують алгоритми напівавтоматичного навчання, що позбавляє від необхідності власноруч створювати величезні об’єми даних. Також зараз існує багато наборів даних у відкритому доступі для навчання систем розпізнавання іменованих сутностей, що також полегшує задачу розробки.

У цілому, подібні системи сьогодні є досить ефективними, наприклад, найкраща система, що займалася розпізнаванням сутностей у MUC-7 мала оцінку F1 93.39%, тоді як оцінка результату роботи людини складала 97.6 % [7].

1.2 Порівняльний аналіз аналогів

Як правило, системи розпізнавань іменованих сутностей є частинами програмного комплексу для обробки природної мови. Серед таких систем найбільш відомими є:

– Microsoft Azure Cognitive Services;
– Google Natural Language;
– Apache OpenNLP;
– GATE.

Microsoft Azure Cognitive Services – набір веб API для роботи з алгоритмами машинного навчання [8]. До складу цих сервісів входять сервіси для роботи з природною мовою, у тому числі й функція пошуку іменованих сутностей (рисунок 1.1).

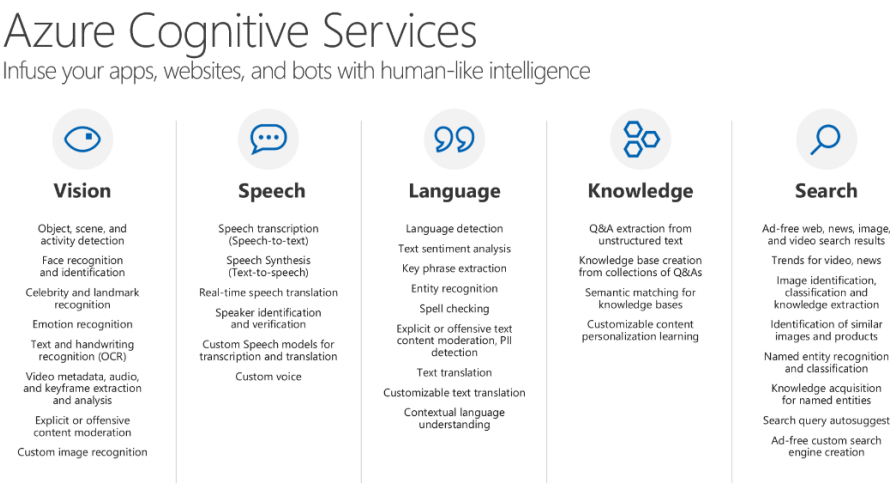

Рисунок 1.1 – Microsoft Azure Cognitive Services

До переваг сервісу відноситься висока точність розпізнавання, швидкодія, підтримка багатьох мов. До недоліків можна віднести високу вартість.

Google Natural Language – набір веб API від компанії Google для роботи з природною мовою [9]. Як і Azure Cognitive Service є платним сервісом, доступ до якого здійснюється через Web API. До переваг сервісу відноситься висока точність розпізнавання, швидкодія, підтримка багатьох мов. До недоліків можна віднести високу вартість.

Apache OpenNLP – бібліотека інструментів, які використовують машинне навчання для обробки природної мови [10]. Підтримує найпоширеніші завдання обробки природної мови, такі як: визначення мови, токенізація, сегментація речень, розмічування частин мови, розпізнавання іменованих сутностей, поверхневий аналіз та синтаксичний аналіз.

General Architecture for Text Engineering або GATE – це набір інструментів Java, спочатку розроблений у Шеффілдському університеті [11]. Починаючи з 1995 року широко використовується для багатьох завдань з обробки природних мов. Інтерфейс додатку зображено на рисунку 1.2.

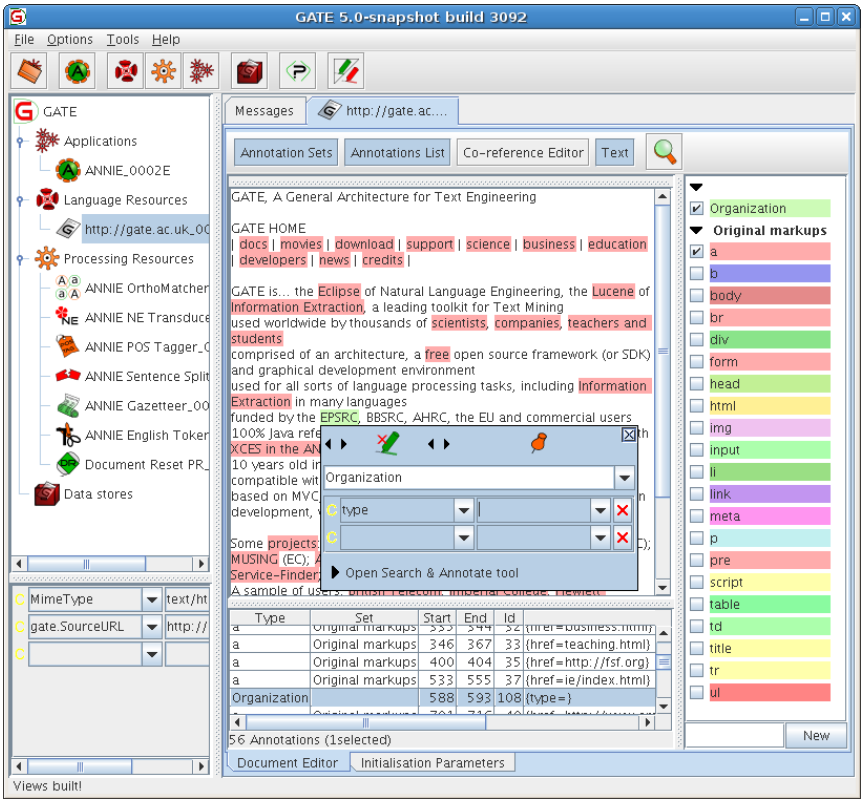

Рисунок 1.2 – Інтерфейс програми GATE

Результати порівняння власного рішення з аналогами наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Apache OpenNLP	GATE	Власний додаток
Стійкість до помилок	-	-	+
Безкоштовна ліцензія	+	+	+
Можливість інтеграції без підключення до мережі інтернет	+	+	+
Підтримка декількох мов	+	-	-
Приблизна оцінка ефективності F1	77%	78%	93-94%

Приблизна оцінка ефективності роботи була отримана на основі оцінок існуючих рішень, які використовують схожі підходи.

Таблиця порівняльних характеристик показала, що розробка програмного продукту є доцільною. В результаті отримаємо продукт, що покриває недоліки існуючих рішень та забезпечує кращу, у порівнянні з безкоштовними аналогами, ефективність.

1.3 Аналіз методів розв’язання поставленої задачі

Варіантів для розв’язання задачі з пошуку іменованих сутностей у тексті є декілька. Одним із перших та найбільш очевидних варіантів є створення словника, у який поміщаються усі слова та їх спільнокореневі варіанти. Далі програма намагається найти у тексті слова, подібні до тих, які є у словнику. Даний підхід є досить неефективним, так як потребує наявності величезної бази слів та великої кількості системних ресурсів, адже необхідно перевірити на

наявність у тексті кожне слово із словника. Через ці проблеми, даний метод майже не використовується.

Більш ефективними є методи, які використовують у інших задачах обробки природної мови. Наприклад, за допомогою методів синтаксичного аналізу тексту, можна знайти більшість іменованих сутностей, особливо, коли мова йде про власні назви, такі як імена людей, компанії, країн, міст, тощо. Даний метод є більш ефективним, тому його використовували частіше.

Найпоширенішим варіантом розв’язування задачі розпізнавання іменованих сутностей у тексті є використання нейронних мереж. Завдяки можливості будувати нейронні мережі з різною архітектурою, починаючи від звичайних класифікаторів, закінчуючи складними рекурентними нейронними мережами, можна підлаштовувати алгоритм під більш конкретні випадки. Завдяки високій ефективності та гнучкості метод з використанням нейронних мереж є найбільш поширеним.

Для вирішення питання розпізнавання іменованих сутностей було обрано метод з використанням нейронних мереж, адже він забезпечить високу ефективність та надійність роботи програмного додатку.

1.4 Аналіз методів NER з використанням нейронних мереж

Штучна нейронна мережа – це комп’ютерна система, яка за принципом своєї роботи схожа на біологічну нейронну мережу, тобто мозок. Така мережа складається з великої кількості вузлів, які є аналогами нейронів у мозку. Вузли поєднуються між собою штучним аналогом синапсів, що дозволяє їм обмінюватися інформаціє один між одним. Такі системи «навчаються» виконувати поставлену задачу на основі навчальних даних. Корекція результату відбувається за рахунок корекції ваг ребер, які з’єднують нейрони.

Базові штучні нейронні мережі, як правило, складаються з 3 шарів:

– шар входу;
– прихований шар;

– шар виходу.

Загальне представлення такої мережі зображено на рисунку 1.3.

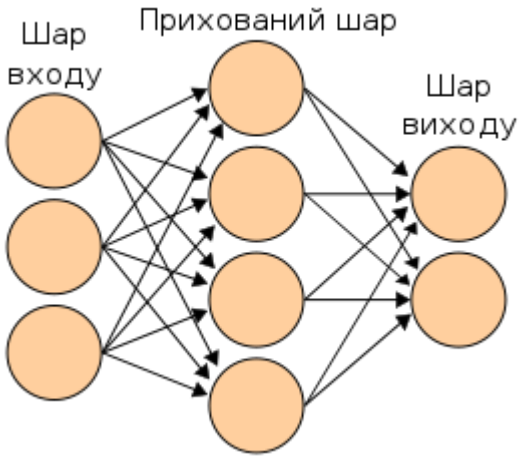

Рисунок 1.3 – Архітектура базової нейронної мережі

Як правило, у нейронних мережах кожна вершина поєднана з усіма вершинами наступного шару. У процесі навчання мережа корегує ваги ребер між вершинами. Це дозволяє для кожної характеристики встановити міру її впливу на ймовірність того чи іншого результату.

Прості нейронні мережі використовують для базових задач класифікації. Наприклад, коли за заданими параметрами необхідно щось передбачити. Для більш складних випадків, таки як класифікація зображень, обробка природної мови, тощо, використовують глибинні нейронні мережі. Головною відмінністю глибинних нейронних мереж від звичайних є те, що вони мають більше одного прихованого шару, що дозволяє апроксимувати значно складніші функції.

Серед глибинних нейронних мереж виділяють два основних види:

– згорткові нейронні мережі (з англ. Convolutional Neural Network або CNN);
– рекурентні нейронні мережі (з англ. Recurrent Neural Network або RNN).

Згорткові нейронні мережі – це клас глибинних нейронних мереж, які широко використовуються у задачах обробки зображень [12]. CNN складаються з трьох типів шарів:

– convolutional layer (шар згортки);
– pooling layer (шар об’єднання);
– fully-connected layer (шар повного об’єднання).

Приклад архітектури згорткової нейронної мережі наведено на рисунку 1.4.

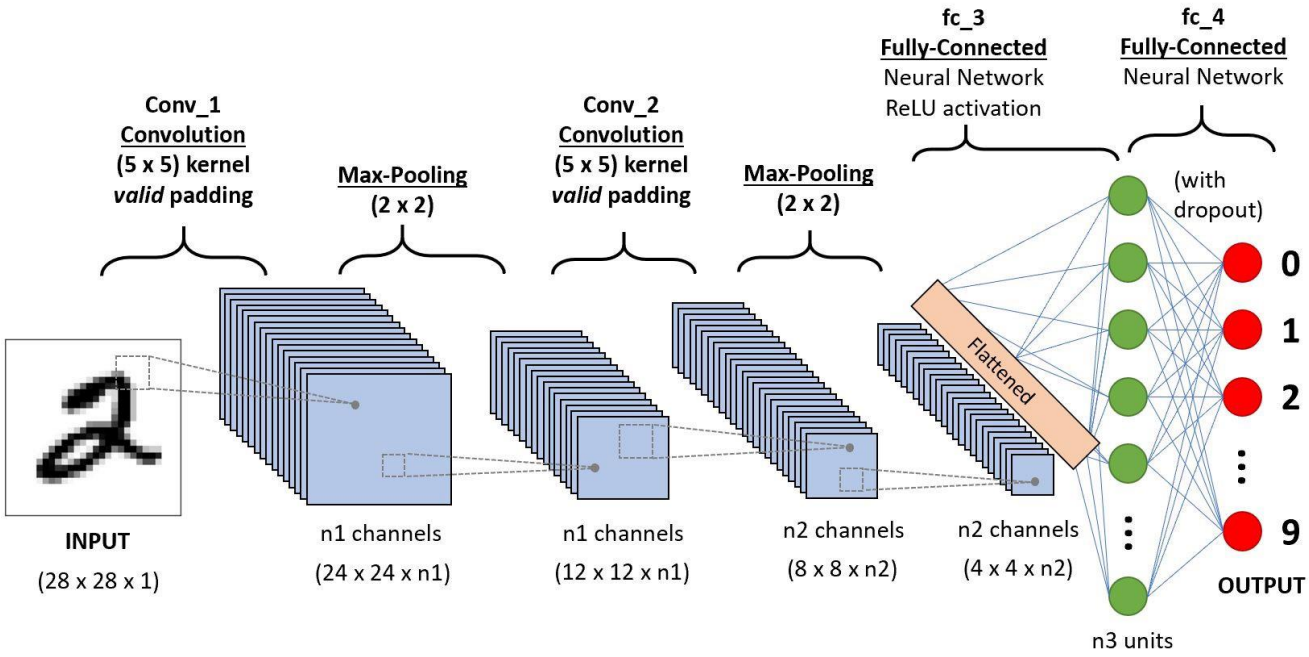

Рисунок 1.4 – Приклад згорткової нейронної мережі

Шар згортки аналізує вхідні дані за допомогою набору фільтрів. У результаті аналізу отримуємо набір особливостей, які подані у вигляді n-мірного масиву. Далі цей масив направляється у шар об’єднання, який зменшує кількість особливостей шляхом певних операцій над масивом. У результаті матимемо зжатий набір особливостей. Це необхідно для того, щоб пришвидшити роботу алгоритму. Послідовність із шарів згортки та об’єднання повторюється декілька раз, у залежності від виду нейронної мережі. Після цього остаточний набір властивостей надсилається до шару повного об’єднання. На цьому етапі проводиться аналіз усіх властивостей, отриманих у ході роботи нейронної мережі та визначається фінальний результат.

Рекурентні нейронні мережі – це клас нейронних мереж які дозволяють аналізувати вхідні дані в контексті попередніх або наступних даних. Такий вид

нейронних мереж є особливо ефективним при аналізі зв’язаних послідовностей, таких як текст, відео, аудіо, тощо [13]. Загальна архітектура таких мереж зображена на рисунку 1.5.

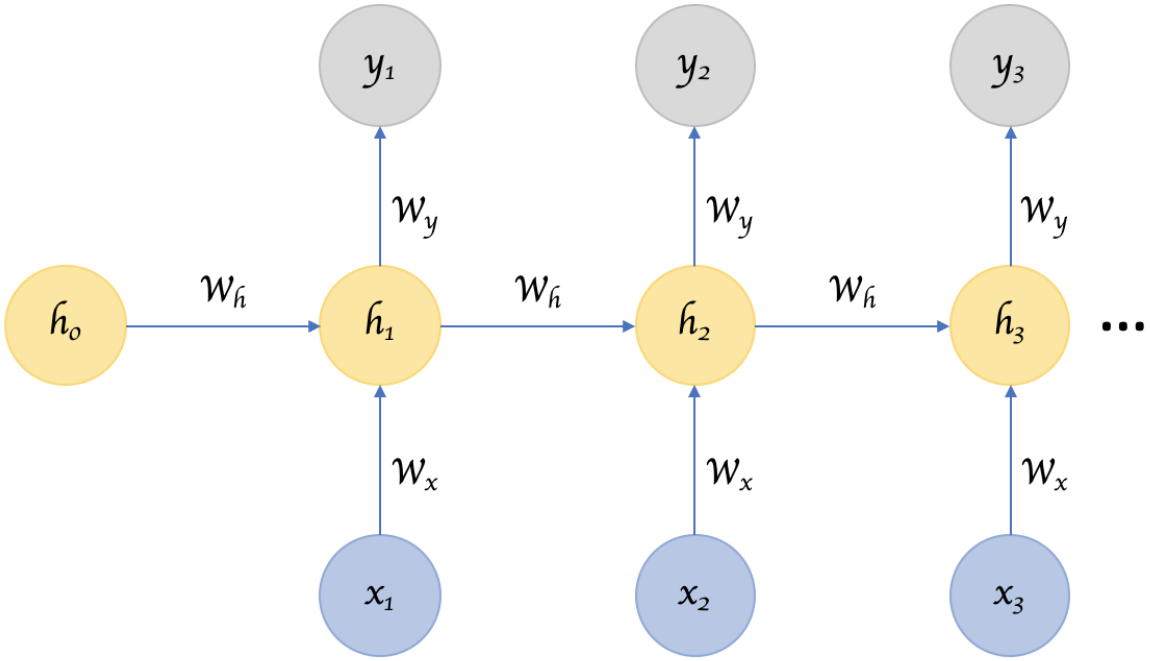

Рисунок 1.5 – Архітектура рекурентної нейронної мережі

Нейронні мережі зазвичай приймають на вхід послідовність даних фіксованого розміру, що обмежує їх використання у задачах, коли необхідно опрацьовувати послідовності невизначених заздалегідь розмірів, наприклад, текст. Рекурентні нейронні мережі ж розроблені так, що можуть приймати на послідовності не визначеного заздалегідь розміру.

На перший погляд, можна використати звичайну мережу декілька разів, залежно від розміру послідовності, але у такому випадку втрачається контекст послідовності. Рекурентні мережі розроблені так, що дозволяють опрацьовувати послідовності будь-яких розмірів і при цьому зберігати контекст. Мережа запам’ятовує інформацію про попередні результати і використовує її для того, щоб передбачити наступний результат, таким чином послідовність подачі даних відіграє велику роль у роботі таких мереж.

Але бувають випадки, коли не тільки попередні дані мають вплив на поточні, але й наступні також. Наприклад, наприкінці речення є слово, яке значною мірою змінює значення декількох слів, що були перед ним. Для таких випадків використовують двонаправлені рекурентні нейронні мережі. Архітектура такої мережі зображена на рисунку 1.6.

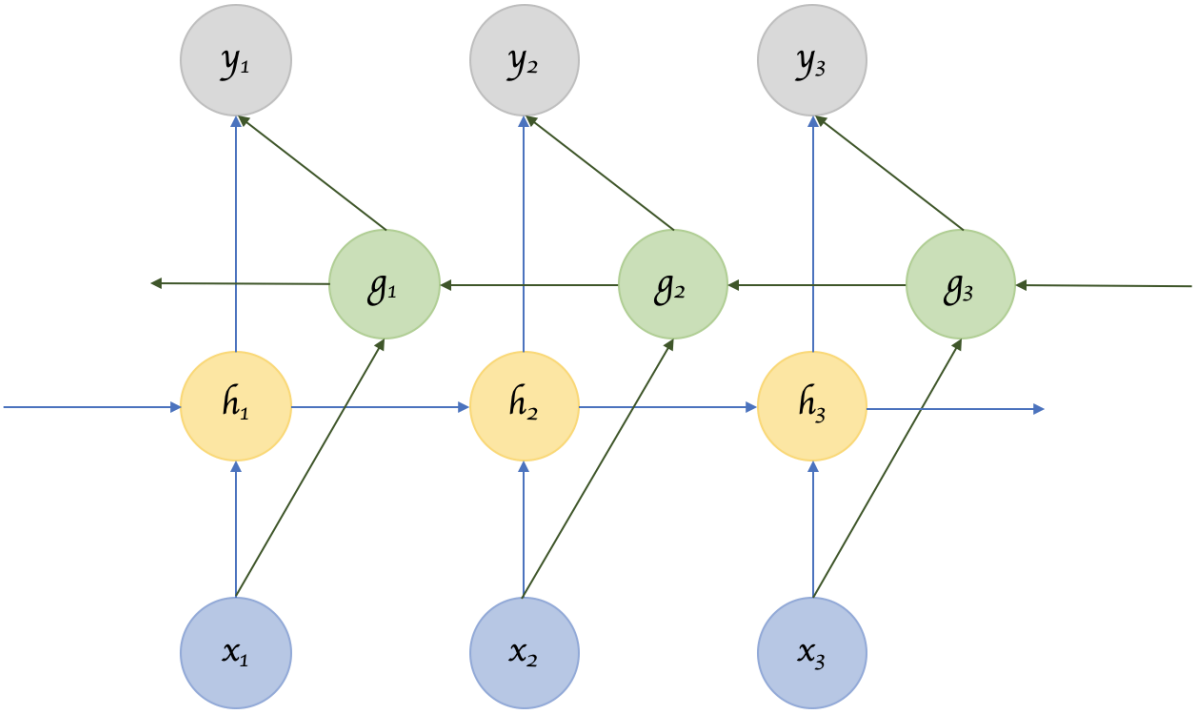

Рисунок 1.6 – Архітектура двонаправленої рекурентної нейронної мережі

Особливістю двонаправлених рекурентних мереж є те, що вони аналізують послідовність в обидві сторони. Як правило, для задач обробки текстової інформації використовують нейронні мережі саме такої архітектури.

Хоча й згорткові та рекурентні мережі можна використати для задачі розпізнавання іменованих сутностей, але структура двонаправленої рекурентної нейронної мережі є найбільш оптимальною для цієї задачі, адже дозволяє аналізувати окремі слова у контексті всього речення. Тому для розробки програмного модуля було обрано алгоритм, який базується на використанні двонаправленої рекурентної нейронної мережі.

1.5 Постановка задачі для розпізнавання іменованих сутностей у тексті

Проаналізувавши питання розробки систем для розпізнавання іменованих сутностей у тексті, було визначено завдання, які необхідно виконати для розробки програмного додатку:

- визначити найбільш ефективний підхід до пошуку іменованих сутностей у тексті;
- розробити метод підготовки вхідних даних;
- розробити метод пошуку іменованих сутностей у тексті;
- розробити модуль пошуку іменованих сутностей;
- розробити веб-сервіс для роботи з модулем пошуку іменованих сутностей;
- провести тестування програмного додатку.

1.6 Висновки

У першому розділі було розглянуто стан питання розпізнавання іменованих сутностей у тексті на сьогоднішній день.

Проаналізовано існуючі аналоги та проведено їх порівняння з розроблюваним програмним продуктом. У результаті порівняння було доведено доцільність розробки власного програмного додатку.

Проведено аналіз існуючих підходів до вирішення поставленої задачі. У результаті аналізу було обрано алгоритм на основі двонаправленої рекурентної нейронної мережі для вирішення задачі.

Сформульовано основні завдання, які необхідно виконати для розробки програмного додатку.

2 РОЗРОБКА МЕТОДІВ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ ПОШУКУ ІМЕНОВАНИХ СУТНОСТЕЙ

2.1 Аналіз моделей мови

Будь-який програмний засіб тим чи іншим чином працює з даними. Як правило, додаток приймає на вхід дані у певному форматі, опрацьовує їх та видає у зрозумілому для користувача вигляді. У задачах обробки природної мови, частіше за все, вхідні дані – це текст, тому його необхідно перетворити в зрозумілий для комп'ютера формат.

Існує декілька підходів до перетворення тексту на числові дані. Найпростіший спосіб – кодування з використанням унітарного коду. Для цього методу необхідно проаналізувати весь вхідний текст, скласти список усіх слів і для кожного із слів призначити унітарний код. У результаті вхідний текст буде перетворено у матрицю, яка може бути подана на вхід до нейронної мережі. Такий спосіб є досить простим в реалізації, але малоефективним.

Одним із недоліків кодування з використанням унітарного коду є те, що він дає інформацію тільки про наявність слова у тексті або реченні, тому, як правило використовують інші, більш досконалі, методи кодування слів, одним із яких є метод під назвою «торба слів» (англ. Bag-of-words). Даний метод допомагає отримати репрезентацію тексту, що має інформацію про слова та частоту їх використання. Особливістю цього методу є те, що він описує весь текст у загальному, а не кожне слово окремо, тому його часто використовують у задачах класифікації документів.

Щоб використовувати цей метод, перш за все, необхідно мати словник. Далі текст розбивається на слова й рахується кількість кожного із них окремо. У результаті ми отримуємо масив, у якому напроти кожного індексу слова буде частота його використання у тексті. Частота використання слова може бути подана як у абсолютних значеннях, так і у відносних. Даний метод можна

використовувати і для аналізу кожного речення тексту окремо, що може значно покращити надійність. Приклад методу зображено на рисунку 2.1.

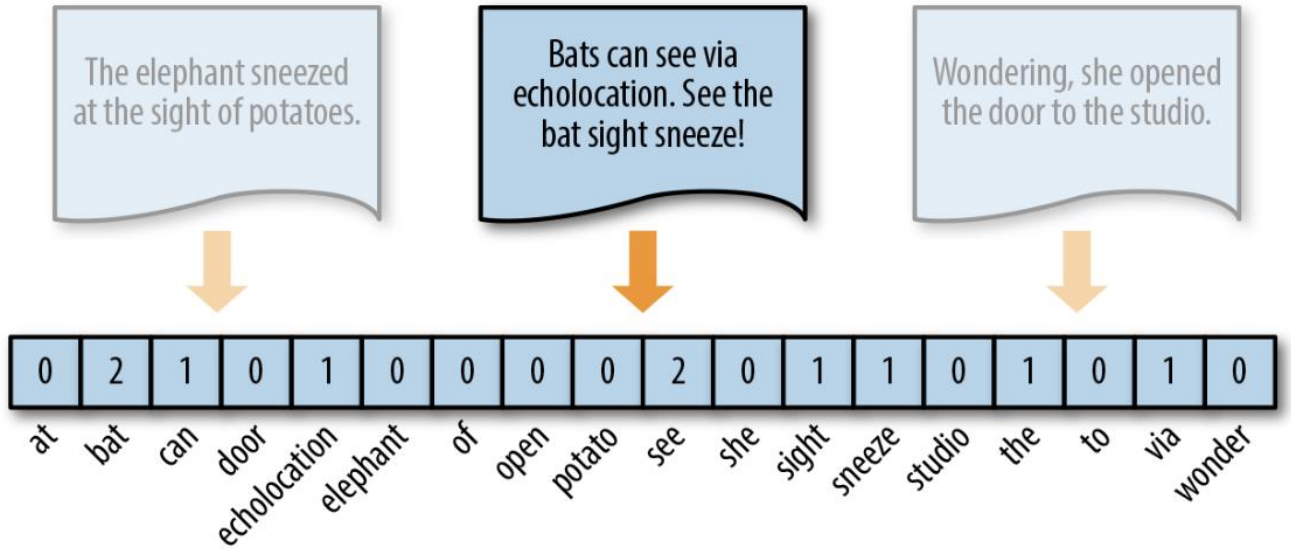

Рисунок 2.1 – Метод bag-of-words

Хоча bag-of-words є простим і досить ефективним методом перетворення текстових даних у числові, але він має ряд недоліків, що не дозволяють використовувати його у повній мірі в задачах розпізнавання сутностей. Одним із таких недоліків є те, що метод дає інформацію про весь текст або речення в цілому, через що відсутня інформація про позицію кожного слова у тексті.

На сьогоднішній день у задачах обробки природної мови найчастіше використовують метод векторного представлення слів (з англ. Word embeddings). Даний метод дозволяє отримати представлення слова у вигляді n-мірного вектору, у якому кожне значення відповідає тій чи іншій характеристиці слова [14]. Даний метод є найбільш досконалим, адже дозволяє отримати семантичне значення окремого слова, що значно покращує надійність роботи алгоритму, який його використовує.

Представлення слів отримують за допомогою алгоритмів машинного навчання, що аналізують використання слів у тексті, що дозволяє враховувати

синоніми та близькі за значенням слова. Числові представлення таких слів матимуть близькі за значенням вектори. Приклад методу зображено на рисунку 2.2.

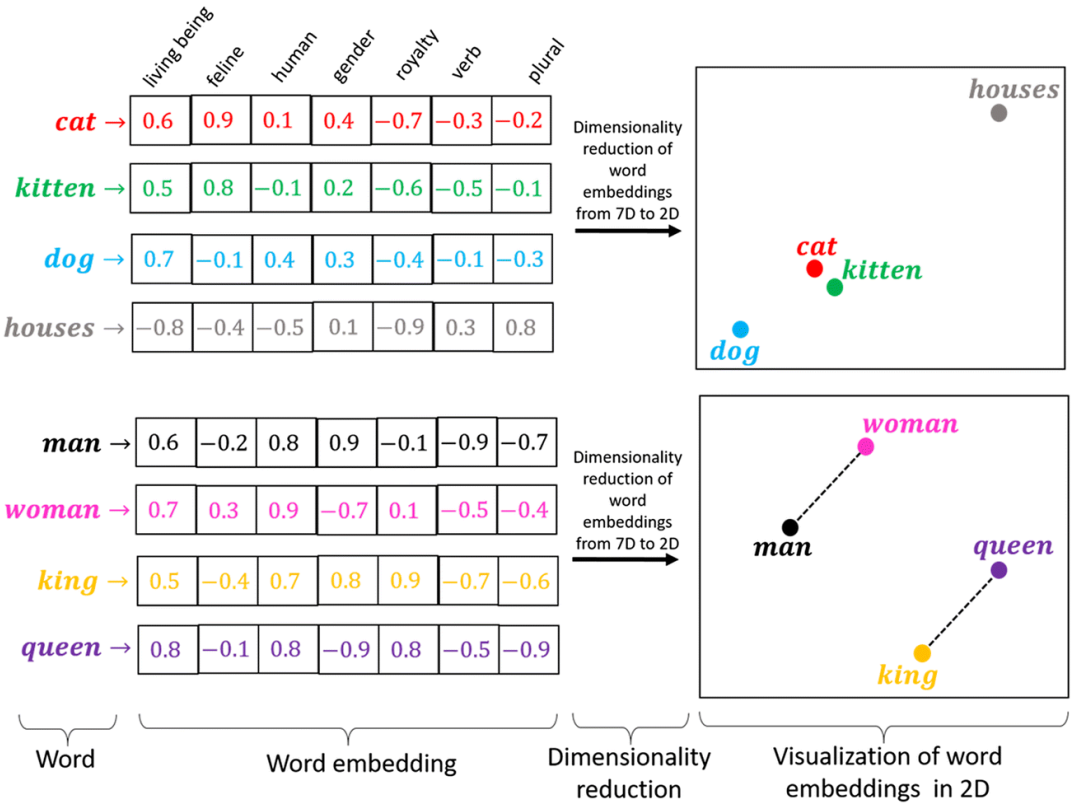

Рисунок 2.2 – Метод word embedding

Процес навчання може бути пов’язаним з конкретною задачею обробки природної мови або ж бути незалежним і використовувати інший набір даних. Серед незалежних алгоритмів навчання найчастіше використовують Word2Vec та GloVe, адже існують вже навчені моделі на основі цих алгоритмів.

Останнім часом на зміну word embedding приходять більш складні моделі, які аналізують значення слова в контексті фрази або цілого речення. Однією із передових моделей мови на сьогоднішній день є BERT. Ця модель мови розроблена компанією Google, в першу чергу для пошукових систем. Ключова особливість моделі BERT полягає в аналізі слова в контексті всього речення, таким чином, вектор слова буде змінюватись в залежності від того, як саме слово використовується у реченні.

Ще однією особливістю BERT є процес аналізу невідомих слів, тобто таких, яких немає у словнику моделі. Для цього модель працює з частинами слів, які зберігаються у спеціальному словнику. Якщо на вході буде невідоме слово, то воно буде розбито на кілька складових, які вже можуть бути відомі моделі. Дани метод не до кінця вирішує проблему невідомих слів, адже все ще можливо отримати таку частину, якої немає у словнику моделі.

Одним із варіантів BERT є CharacterBERT (рисунок 2.4). Головна відмінність даної моделі від BERT полягає в тому, що вона відмовляється від використання словника частин слів, а використовує CharacterCNN модуль для первинного аналізу слова. Це дозволяє моделі працювати з будь-якими вхідними даними. Також такий підхід дозволяє отримувати більш точні результати при аналізі тексту з помилками, що позитивно впливає на надійність [15].

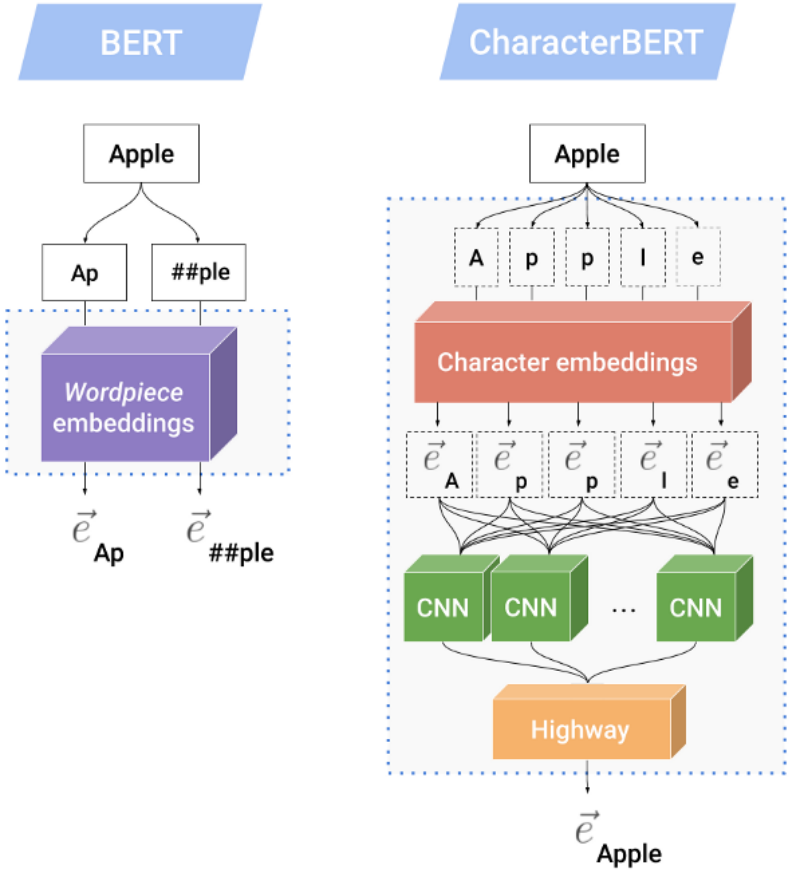

Рисунок 2.4 – Порівняння моделей BERT та CharecterBERT

Використання CharacterBERT в якості моделі мови для задачі пошуку іменованих сутностей у тексті є найбільш доцільним, адже це дозволить отримувати точне семантичне значення слова, за рахунок того, що модель аналізує слово в контексті, а також підвищить надійність системи.

2.2 Розробка алгоритмів роботи системи

Першим етапом розробки системи є проектування її загального алгоритму роботи. Розроблювана система працюватиме як веб-сервіс. Таким чином взаємодія із системою буде здійснюватися шляхом надсилання HTTP запитів на сервер, який, у свою чергу, працюватиме з нейронною мережею. Загальний алгоритм роботи системи зображено на рисунку 2.5.

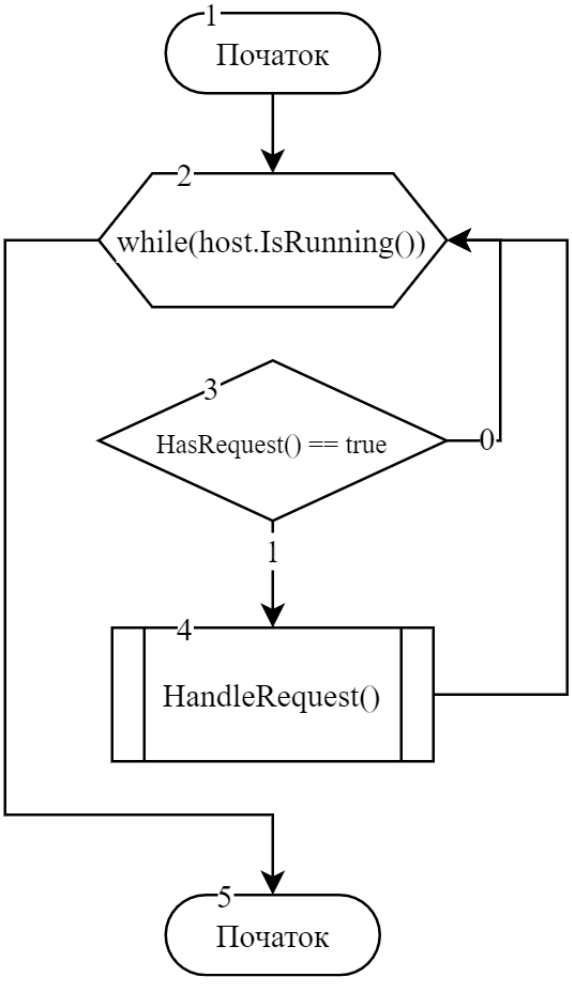

Рисунок 2.5 – Загальний алгоритм роботи системи

Наступний етап – розробка алгоритму обробки запитів. Першим етапом алгоритму є отримання вхідних даних із запиту. Далі необхідно виконати обробку даних перед передачею їх до мережі на аналіз. Наступний етап – обробка даних мережею. Дані, отримані від мережі також необхідно перетворити у більш зручний для сприйняття формат. Останній етап – надсилання відповіді на запит. Блок-схема алгоритму зображена на рисунку 2.6.

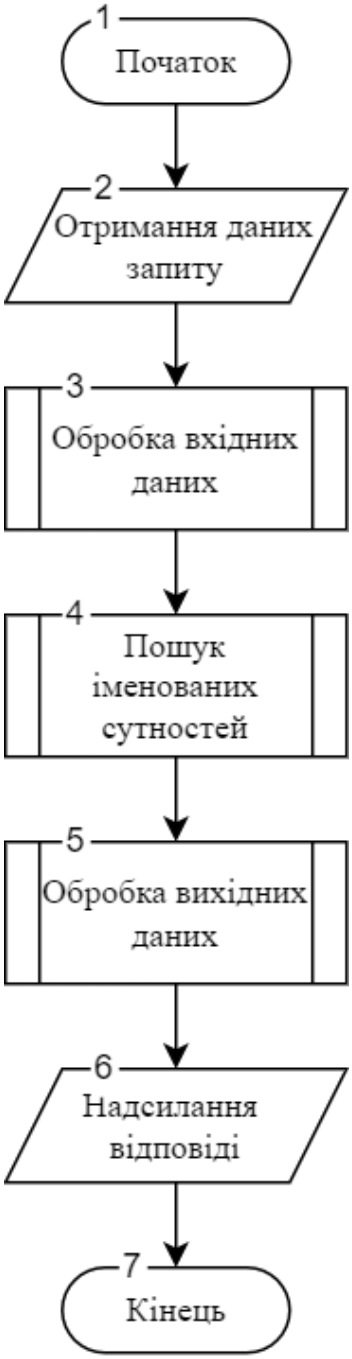

Рисунок 2.6 – Блок-схема алгоритму обробки запиту

На виході з нейронної мережі маємо масив векторів. Кожен елемент вектору – це ймовірність того, що слово належить до певної категорії. Обравши максимальний елемент вектору, отримаємо індекс класу, до якого належить слово. Блок-схема алгоритму зображена на рисунку 2.7.

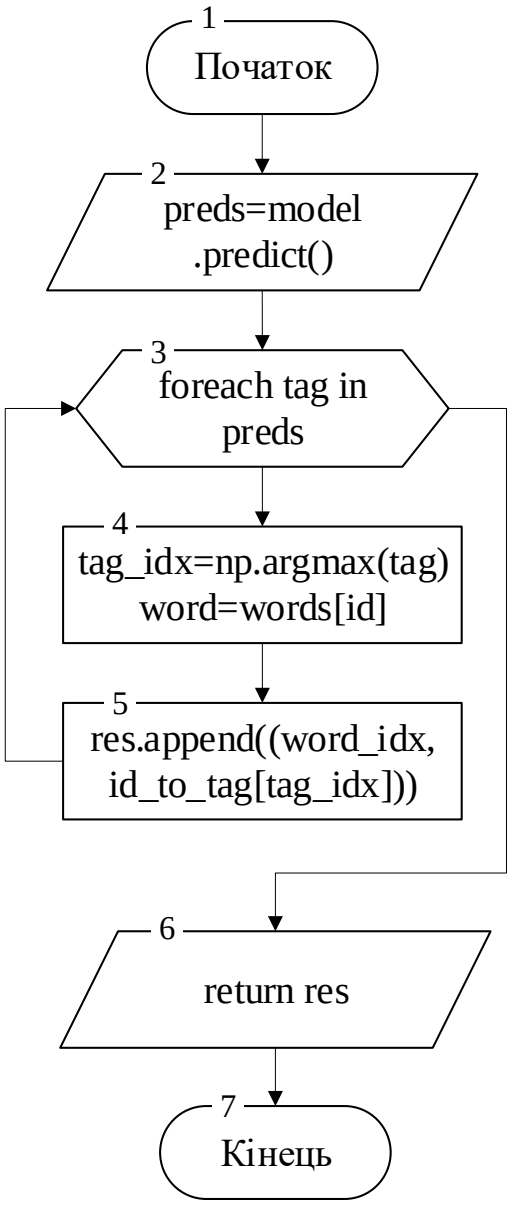

Рисунок 2.7 – Блок-схема алгоритму перетворення вихідних даних

2.3 Розробка методу підготовки вхідних даних

Через особливості роботи нейронних мереж вхідні дані необхідно правильно підготувати перед тим, як передавати у мережу на обробку.

У задачі пошуку іменованих сутностей вхідними даними є текст. Оскільки мережа не вміє працювати з текстовими даними напряму, то даний текст необхідно перетворити в масив чисел. Блок-схема алгоритму зображена на рисунку 2.8.

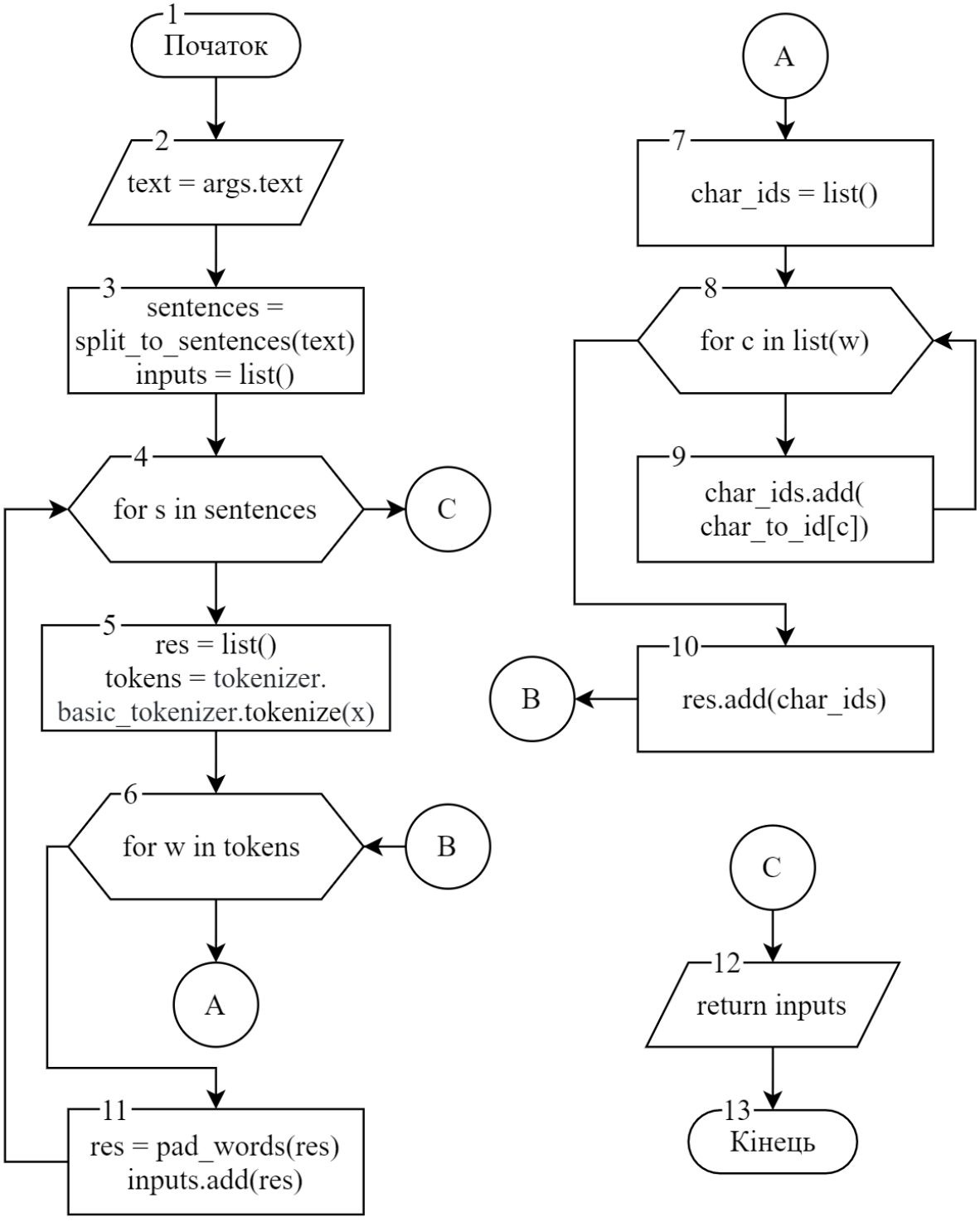

Рисунок 2.8 – Блок-схема алгоритму обробки вхідних даних

Основні етапи методу обробки вхідних дані наступні:

1. Отриманий текст розбивається на окремі речення.
2. Кожне із речень розбивається на окремі токени.
3. Усі токени вирівнюються по довжині за рахунок додавання спеціального символу в кінець.
4. Усі токени розбиваються на окремі символи.
5. Для кожного символу визначається його порядковий номер у словнику.
6. Усі масиви порядкових номерів формуються у двовимірний масив, що являє собою представлення речення для нейронної мережі.

Першим етапом обробки є розподіл тексту на окремі речення. Даний етап не є обов'язковим, але дозволяє підвищити надійність роботи системи. Цей етап є доцільним, адже більшість наборів навчальних даних для Named Entity Recognition містять у собі окремі речення, а не тексти, через що мережа навчається розуміти та шукати сутності в межах одного речення.

Далі необхідно розбити речення на токени (цей процес, як правило, називають токенізацією). Токен являє собою одиницю мови для нейронної мережі, це може бути слово, знак пунктуації, певні додаткові символи для мережі, тощо. Для прикладу, речення «Hello World!» у результаті буде розбите на наступні токени: «hello», «world» та «!». Також для моделей мови BERT та CharacterBERT додатково додаються токени початку та кінця послідовності «[CLS]» та «[SEP]».

Наступний крок – всі токени у послідовності необхідно розбити на окремі символи, адже модель мови CharacterBERT використовує символьний аналіз слова, а не аналіз частин.

Так як для кожного символу мережі потрібно отримати його вектор зі словника, то замість символів необхідно подавати порядковий номер символу в словнику.

Важливим моментом є те, що при роботі з масивами мережа може працювати лише з масивами однакових розмірів, наприклад, якщо є слова «table»

та «sofa», то масиви символів слів будуть мати різний розмір, що не є допустимим. Тому необхідно зробити так, щоб всі масиви символів мали однакову довжину. Для цього визначається найдовший масив, а до менших масивів додаються спеціальні символи. При навчанні мережа розуміє, що дані символи необхідно ігнорувати, тому вони не матимуть впливу на кінцевий результат. Виконувати дану операцію над реченнями не потрібно, адже речення подаватимуться до мережі окремо одне від одного.

У результаті роботи алгоритму з вхідного тексу буде отримано тривимірний масив «inputs», що міститиме у собі масиви індексів слів для кожного слова у всіх реченнях тексту.

Окрім порядкових номерів символів на вхід моделі мови, що базуються на BERT, додатково потребують масив сегментів та маску уваги (attention mask).

Масив сегментів використовується в BERT для розподілу тексту на певні сегменти, тобто речення. Розмір цього масиву має дорівнювати кількості токенів у вхідному реченні. Масив складається з номерів речень у тексті, наприклад, токену, що належить першому реченню буде назначено індекс «0», а для другого – «1». Для задачі пошуку іменованих сутностей використання сегментації не є доцільним, адже мережі, як правило, навчаються знаходити іменовані сутності в контексті одного речення. Таким чином, масив сегментів буде складатися лише з індексів «0».

Маска уваги використовується для того, щоб повідомити BERT, які саме токени варто брати до уваги при формуванні векторних представлень. Така необхідність виникає через те, що дані в мережу можуть подаватися партіями, щоб прискорити процес навчання. У такому випадку, необхідно всі речення у партії вирівнювати в одну довжину. Це робиться за рахунок додавання спеціальних токенів у кінець речень. Такі токени необхідно ігнорувати, щоб підвищити точність формування векторних представлень. Маска уваги складається з індексів «0» та «1». Токени, що беруть участь у формуванні векторних представлень, помічаються індексом «1», а ті, що потрібно ігнорувати,

індексом «0». Оскільки для передбачення на вхід до мережі подається одне речення, то маска уваги може складатися лише з «1».

Таким чином додатково маємо масив сегментів та маску уваги, які необхідно подати на вхід до мережі

2.4 Розробка методу пошуку іменованих сутностей

Метод пошуку іменованих сутностей базуватиметься на використанні нейронних мереж, тому необхідно розробити її архітектуру.

Оскільки задачею нейронної мережі є аналіз текстових даних, то за основу буде взято двонаправлену рекурентну нейронну мережу [16].

Так як метод пошуку іменованих сутностей базується на використанні нейронної мережі, то його етапи – це шари нейронної мережі. Нейронна мережа складається з наступних частин:

1. Вхідні дані. На вхід мережа приймає тривимірний масив (розмір партії, максимальна кількість токенів у реченні, довжина максимального слова у реченні), що містить порядкові номера символів слів у реченні (input_ids), маску уваги (attention_mask) та масив сегментів (segment_ids).
2. Шар CharacterBERT. Даний шар відповідає за формування векторних представлень слів. За рахунок використання CharacterBERT, а не будь-якої іншої моделі мови, вдається підвищити надійність пошуку іменованих сутностей, адже модель мови CharacterBERT є більш стійкою до помилок.
3. Шар виключення (Dropout). Використовується для регуляризації нейронної мережі.
4. Шар BiLSTM. Використовується для двонаправленого аналізу векторів слів, на основі якого формуються певні логічні зв’язки між словами у реченні.
5. Шар Linear. Зводить вихід з BiLSTM шару у тривимірний тензор (розмір партії, максимальна кількість токенів у реченні, кількість класів).
6. Шар CRF. Відповідає за остаточну класифікацію слів у реченні.

Першим етапом побудови архітектури нейронної мережі є визначення вхідних та вихідних даних. На вході мережа приймає двовимірний масив індексів символів слів, маску уваги та масив сегментів. На виході – матриця ймовірностей, де для кожного слова з вхідних даних матимемо масив, у якому записані ймовірності по кожному із класів.

Після вхідного шару йде шар моделі мови. Його основна задача полягає у тому, щоб з вхідних даних (масивів символів) створити вектори слів, які будуть використані мережею у подальшому аналізі.

Для покращення надійності роботи системи у якості моделі мови використовується CharacterBERT. Дана модель мови опирається на аналіз символів слова, а не частин слова, як її оригінальна версія BERT. Це дозволяє зробити систему більш стійкою до помилок у тексті, що покращує її надійність.

Далі, отримані вектори подати на вхід до двонаправленої рекурентної мережі.

Для покращення надійності роботи алгоритму було використано архітектуру рекурентної нейронної мережі з використанням довгої короткочасної пам’яті (з англ. Long Short-Term Memory або LSTM). Long Short-Term Memory – вид рекурентних нейронних мереж, особливістю якого є те, що вузли такої архітектури можуть запам’ятовувати значення для довгих або коротких проміжків часу [17].

Звичайні рекурентні нейронні мережі замість того, щоб добавити нову інформацію повністю змінюють поточну, через що вся інформація модифікується в цілому. Таким чином втрачаються дані про те, чи є ця інформація важливою, чи ні. LSTM мережі замість того, що повністю змінити інформацію, лише вносять корективи шляхом множення та додавання. В LSTM мережах інформація проходить через механізм, відомий як стан вершин. Це дозволяє таким мережам вибірково запам’ятовувати або забувати певну інформацію. Архітектуру LSTM мережі зображена на рисунку 2.9.

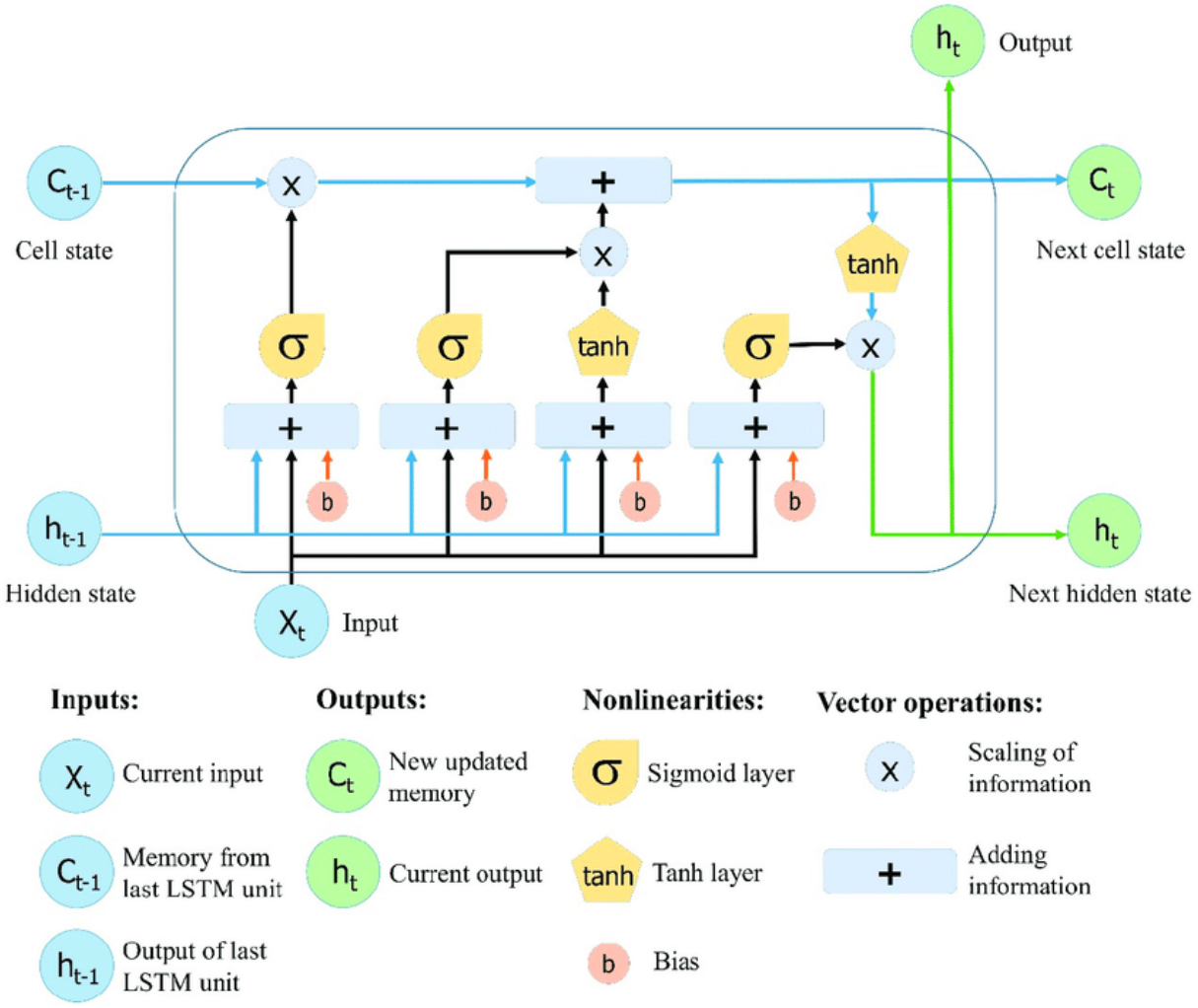

Рисунок 2.9 – Архітектура LSTM мережі

Наступний етап – шар виключення (з англ. Dropout). Цей шар використовується в нейронних мережах для запобіганню їх перенавчанню. Суть полягає в тому, що в процесі навчання мережі випадковим чином виключається певний відсоток нейронів. Кожну нову ітерацію відбувається нова вибірка нейронів для виключення. Це дозволяє значно підвищити швидкість та якість навчання мережі, а також підвищити надійність роботи на нових даних за рахунок регуляризації нейронної мережі. Приклади недовченої, оптимально навченої та перенавченої мережі зображено на рисунку 2.10.

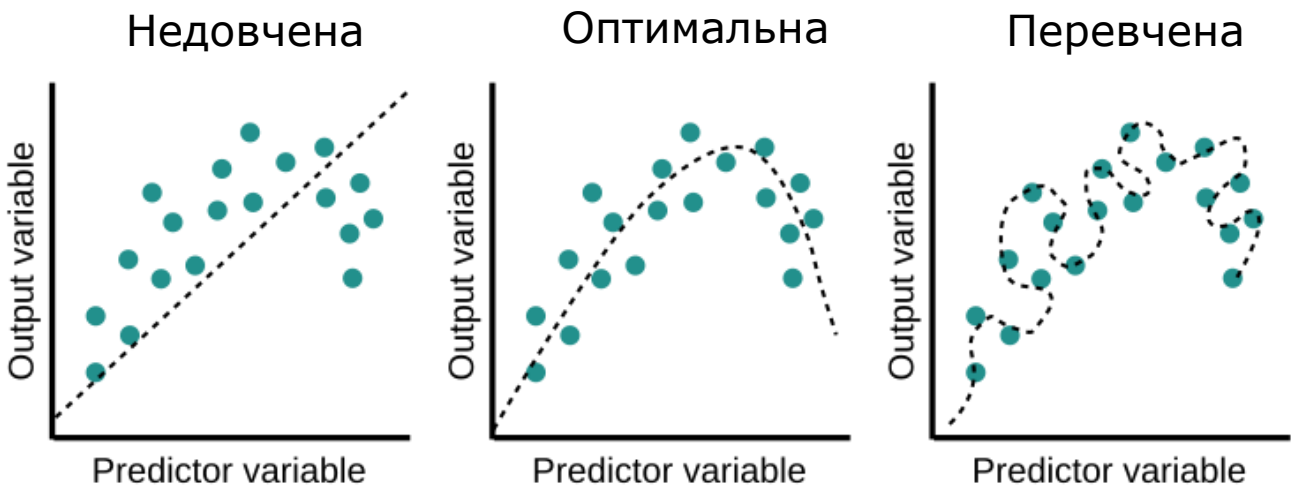

Рисунок 2.10 – Приклади недовченої, оптимально навченої та перенавченої мереж

Останній етап – шар умовного випадкового поля (з англ. Conditional Random Field або CRF). Conditional Random Fields (рисунок 2.11) – клас методів статистичного моделювання, який часто використовують для структурованого передбачення. Як правило, застосовують для передбачення на послідовностях, адже однією із особливостей CRF є те, що даний клас методів може брати до уваги результати сусідніх елементів [18].

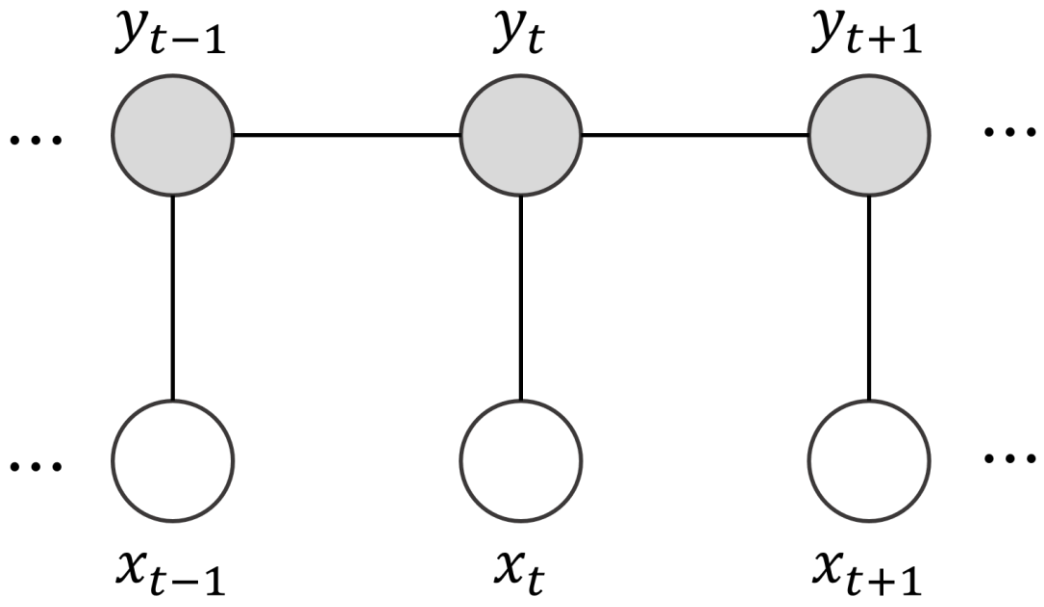

Рисунок 2.11 – Граф моделі шару випадкового умовного поля

У результаті отримаємо CharacterBERT-BiLSTM-CRF нейронну мережу. Структура нейронної мережі зображена на рисунку 2.12.

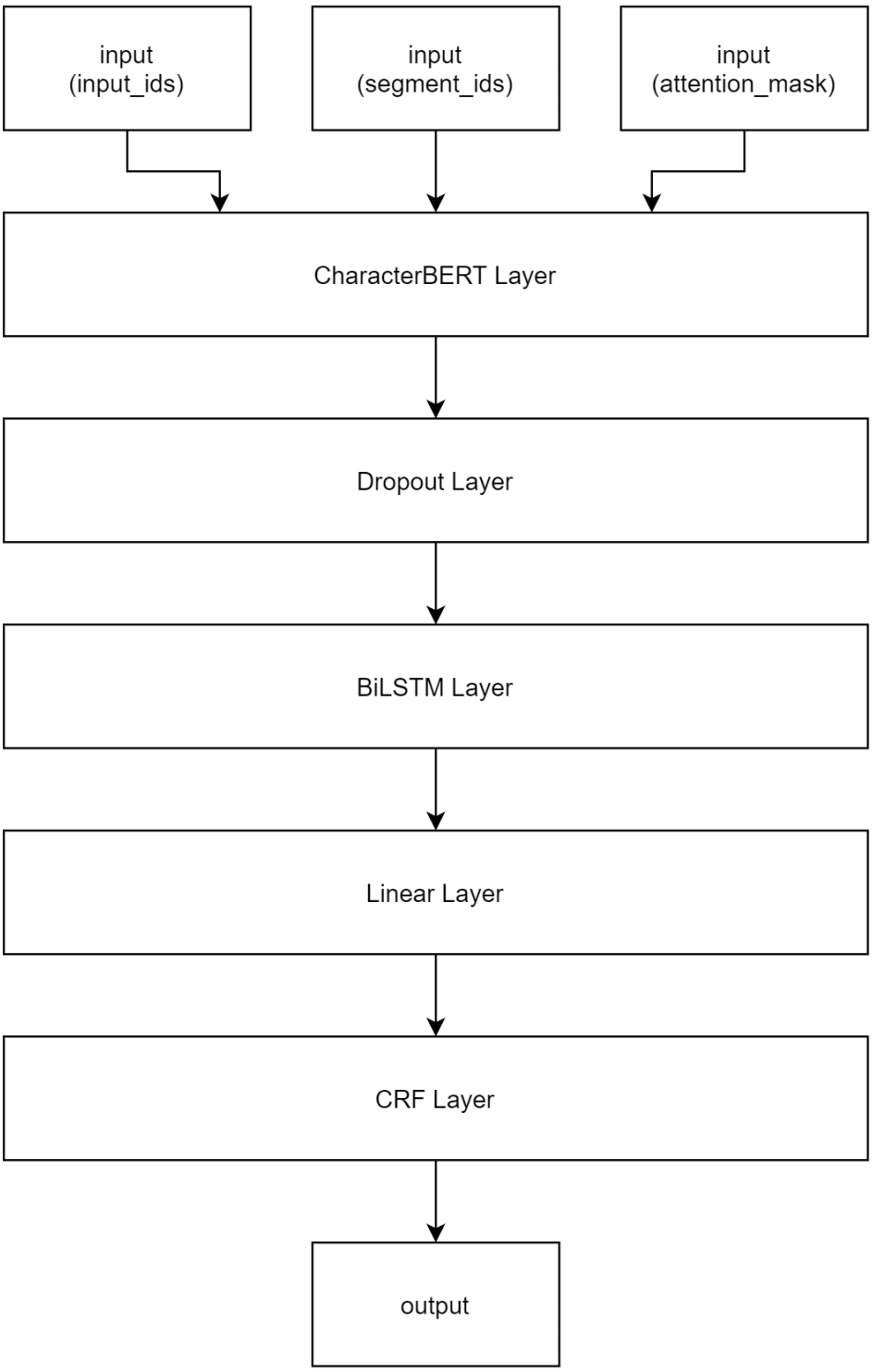

Рисунок 2.12 – Структура нейронної мережі для розпізнавання іменованих сутностей у тексті

2.5 Розробка алгоритму навчання нейронної мережі

Хоча архітектура нейронної мережі має значний вплив на кінцевий результат, але без навчання будь-яка нейронна мережа не буде працювати так, як заплановано. Тому наступним важливим етапом у розробці будь-якої системи на базі нейронних мереж є навчання.

Існує чотири основні підходи до навчання нейронних мереж:

- Навчання з учителем (supervised learning);
- Навчання без учителя (unsupervised learning);
- Напівавтоматичне навчання (semi-supervised learning);
- Навчання з підкріпленням (reinforcement learning).

Навчання з учителем використовують у тих випадках, коли є набір даних для навчання з визначеними результатами. Наприклад, є набір зображень та інформація про те, що і на якому зображенні знаходиться. Таким чином, нейронна мережа має вхідні дані та вихідні. У такому випадку нейронна мережа на основі вхідних даних намагається зробити передбачення, результат якого порівнюється з вихідними даними. У залежності від результатів порівняння відбувається корекція ваг ребер.

При навчанні без учителя в мережі є тільки набір даних, але нема ніяких вказівок про те, що з цими даними робити. У такому випадку мережа намагається сама знайти кореляцію даних за рахунок аналізу. Як правило такий підхід використовують в задачах кластеризації, знаходження аномалій та побудови асоціацій.

Напівавтоматичне навчання є поєднанням навчання без учителя та з учителем. При такому підході у мережі є дані з результатами і без. Цей метод особливо корисний, коли важко добути з даних важливі признаки або розміщення всіх об’єктів є досить трудомісткою задачею. При такому підході досить часто використовуються генеративні змагальні мережі (з англ. Generative Adversarial Network або GAN). Такий метод навчання передбачає наявність двох нейронних мереж, одна з яких є генератором даних, інша ж є класифікатором.

Спочатку класифікатор генератор навчають на невеликому наборі даних, а далі їх об’єднують. На кожній ітерації генератор намагається створити зображення, а класифікатор намагається визначити чи є дане зображення створене генератором. З кожним таким циклом точність обох мереж зростає.

Навчання з підкріпленням по своїй суті сильно схоже на відеогру. Нейронна мережа намагається знайти оптимальний спосіб для досягнення поставленої цілі. Глобальна ціль, у даному випадку – передбачити шаги так, щоб отримати найбільшу «винагороду». Цей метод навчання можна описати на прикладі гри в шахи. Спочатку мережа не буде знати, як грати в шахи і її дії будуть завжди неправильними й приводитимуть до поразок. Після кожної ітерації мережа буде вчитися грати і її дії ставатимуть все більш обдуманими. Це можливо завдяки тому, що мережа має результат своїх дії, у даному випадку це результат партії. Даний алгоритм навчання найчастіше використовують при розробці автомобілів з автономним керуванням або штучного інтелекту в відеоіграх.

Для задачі розпізнавання іменованих сутностей найбільш доцільним є алгоритм навчання з учителем. Це дозволить мати повний контроль над процесом навчання й отримати в результаті мережу, яка має високу надійність.

За сам процес корекції ваг під час навчання в нейронних мережах відповідає оптимізатор. Він оновлює ваги ребер у відповідь на порівняння результату передбачення та правильного результату. Простими словами оптимізатор намагається зробити так, щоб результат роботи мережі якомога точніше відповідав правильному. У цьому йому допомагає функція втрат (loss function), яка покликана відобразити подію або значення кількох величин на дійсне число, яке являє собою певні «витрати», пов’язані з цією подією. Оптимізатор ж намагається зменшити значення цієї функції.

Оптимізація, як правило, виконується з використанням стохастичного градієнтного спуску (з англ. Stochastic Gradient Descent або SGD) та його різновидів. Серед різновидів найбільш популярним є метод Adam (Adaptive

moment estimator). Він поєднує в собі високу ефективність та швидкість навчання, через що й здобув популярність.

Як правило, для підвищення швидкості навчання дані групують партіями. Розмір партії, як правило, залежить від технічних характеристик обладнання, на якому здійснюється навчання. Такий підхід дозволяє проводити обробку декількох незалежних один від одного випадків за раз.

Отже, для навчання нейронної мережі буде відбуватися з учителем. В якості оптимізатора буде використано Adam.

2.6 Висновки

У другому розділі було проведено аналіз моделей мови для нейронних мереж. У результаті чого було вирішено використовувати модель мови CharacterBERT.

Розроблено загальні алгоритми роботи додатку.

Розроблено метод обробки вхідних даних, що на відміну від існуючих розділяє вхідний текст на окремі речення для підвищення надійності пошуму іменованих сутностей

Розроблено метод пошуку іменованих сутностей у тексті з використанням нейронних мереж, який на відміну від існуючих використовує модель мови CharacterBERT.

Обрано метод навчання з учителем та оптимізатор Adam для проведення навчання нейронної мережі.

3 РОЗРОБКА СИСТЕМИ ПОШУКУ ІМЕНОВАНИХ СУТНОСТЕЙ У ТЕКСТІ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу.

Вибір правильних технології для реалізації будь-якого програмного продукту є важливим етапом процесу розробки, адже правильна технологія дозволить значно полегшити процес реалізації. Для розробки додатку було прийнято рішення використовувати дві мови програмування.

Система пошуку іменованих сутностей у тексті буде складатися з двох компонентів: веб-сервіс та нейронна мережа. Для розробки веб-сервісу було обрано мову програмування C#. Це об’єктно-орієнтована мова програмування. Головною особливістю цієї мови є те, що вона тісно пов’язана з платформою .NET Framework, яка містить досить об’ємну стандартну бібліотеку класів, що значно спрощує написання програмного коду [19].

Окрім .NET Framework C# підтримує також .NET Core (або просто .NET, починаючи із версії 5.0). Головною особливістю цієї платформи є те, що на відміну від класичного .NET Framework, .NET Core можна використовувати на різних операційних системах.

Для розробки веб-застосунків за допомогою мови C# та платформи .NET Core використовують фреймворк ASP.NET Core, розроблений компанією Microsoft. ASP.NET Core дозволяє створювати як комплексні веб-застосунки з використанням різни Js фреймворків для клієнтської частини, так і REST API сервіси. Головними перевагами технології ASP.NET Core є:

- наявність активної підтримки від Microsoft;
- підтримка Web API;
- наявність інтегрованого веб-серверу Kestrel;
- підтримка масштабування;
- вища, у порівнянні з популярними аналогами, продуктивність [20].

Для розробки веб-сервісу пошуку іменованих сутностей у тексті доцільно використовувати підхід REST API. Це дозволить зробити процес інтеграції сервісу в будь-які додатки значно простішим.

Враховуючи всі вище перечисленні особливості та переваги, мова C# є найбільш доцільною для розробки веб частини системи пошуку іменованих сутностей.

Python – високорівнева мова програмування загального призначення, яка орієнтована на підвищення продуктивності розробника та простоти коду за рахунок простого синтаксису. Основними особливостями Python є:

- динамічна типізація;
- автоматичне керування пам’яттю;
- механізм обробки виключень;
- підтримка багатопоточних розрахунків;
- високорівневі структури даних;
- кросплатформеність.

Сьогодні Python часто використовують для розробки нейронних мереж, адже його простота та наявність великої кількості сторонніх бібліотек значно полегшують цю задачу. Одними із найпопулярніших бібліотек для розробки нейронних мереж мовою Python є TensorFlow та PyTorch.

TensorFlow – безкоштовна бібліотека з відкритим кодом, розроблена компанією Google для вирішення задач створення та навчання нейронних мереж. Використовується компанією Google як для наукових досліджень в галузях штучного інтелекту так і в комерційних продуктах.

PyTorch – безкоштовна бібліотека з відкритим кодом для машинного навчання, розроблена компанією Facebook. Аналогічно до TensorFlow, ця бібліотека використовується компанією Facebook для розробки внутрішніх алгоритмів, що базуються на нейронних мережах.

PyTorch та TensorFlow у загальному мають схожий функціонал та продуктивність. Це обумовлено тим, що обидві бібліотеки у своїй основі

використовують схожі підходи та механізми. Ключовою відмінністю є те, як саме бібліотеки надають доступ до свого функціоналу. TensorFlow надає доступ до великої кількості елементів низького рівня, тоді як PyTorch працює з абстракціями більш високого рівня [21]. Обидва підходи мають свої переваги та недоліки. Наприклад, підхід, що використовує TensorFlow дозволяє легше будувати нейронну мережу більш детально, тоді як PyTorch дозволяє легко будувати моделі з використанням об'єктно-орієнтованого підходу (torch.nn.Module).

Враховуючи всі переваги та особливості, використання мови програмування Python. У якості бібліотеки для розробки нейронної мережі обрано бібліотеку PyTorch, адже мережа може бути з легкістю побудована з використанням лише високорівневих компонентів.

3.2 Розробка нейронної мережі

Нейронна мережа – основа модуля розпізнавання іменованих сутностей, тому її розробка є найважливішим етапом створення додатку.

Для створення нейронної мережі було використано мову програмування Python та бібліотеку PyTorch.

Для полегшення процесу навчання та використання нейронної мережі було створено окремий клас (BERT_BILSTM_CRF), що буде відповідати за нейронну мережу. Даний клас наслідується від класу BertPretrainedModel, який, у свою чергу від класу – torch.nn.Module, який є базою для моделей нейронних мереж у бібліотеці PyTorch. Ініціалізація основних шарів та деяких параметрів мережі відбувається у конструкторі класу.

Для того, щоб використовувати мережу за допомогою torch.nn.Module необхідно реалізувати метод forward, який приймає вхідні дані до мережі.

Код конструктора класу BERT_BILSTM_CRF зображено на рисунку 3.1, а код методу forward зображено на рисунку 3.2.

```
def __init__(self, config, need_birnn=True, rnn_dim=128):
    super(BERT_BiLSTM_CRF, self).__init__(config)

    self.num_tags = config.num_labels
    self.bert = CharacterBertModel.from_pretrained('./pretrained-models/general_character_bert/', )
    self.dropout = nn.Dropout(0.5)
    out_dim = config.hidden_size
    self.need_birnn = need_birnn

    if need_birnn:
        self.birnn = nn.LSTM(config.hidden_size, rnn_dim, num_layers=1, bidirectional=True, batch_first=True)
        out_dim = rnn_dim*2

    self.hidden2tag = nn.Linear(out_dim, config.num_labels)
    self.crf = CRF(config.num_labels, batch_first=True)
```

Рисунок 3.1 – Код конструктора класу BERT_BILSTM_CRF

```
def forward(self, input_ids, labels, token_type_ids=None, attention_mask=None):
    emissions = self.tag_outputs(input_ids, token_type_ids, attention_mask)
    loss = -1*self.crf(emissions, labels, mask=attention_mask.byte())
    return loss
```

Рисунок 3.2 – Код методу forward класу BERT_BILSTM_CRF

Метод tag_outputs, що відповідає за порядок виконання шарів у мережі зображено на рисунку 3.3.

```
def tag_outputs(self, input_ids, token_type_ids=None, attention_mask=None):

    outputs = self.bert(input_ids, token_type_ids=token_type_ids,
attention_mask=attention_mask)

    sequence_output = outputs[0]

    if self.need_birnn:
        sequence_output, _ = self.birnn(sequence_output)

    sequence_output = self.dropout(sequence_output)
    emissions = self.hidden2tag(sequence_output)

    return emissions
```

Рисунок 3.3 – Код методу tag_outputs класу BERT_BILSTM_CRF

Для отримання результатів передбачення було створено метод predict, код якого зображено на рисунку 3.4.

```
def predict(self, input_ids, token_type_ids=None, attention_mask=None):
    emissions = self.tag_outputs(input_ids, token_type_ids, attention_mask)
    return self.crf.decode(emissions, attention_mask.byte())
```

Рисунок 3.4 – Код методу predict класу BERT_BILSTM_CRF

Для зручнішої роботи з параметрами нейронної мережі та даними було створено клас Config, який відповідає за загальні налаштування модуля. Код класу зображено на рисунку 3.5.

```
class Config():

    path_to_data = './data/conll/'
    output_dir = './result/'

    do_lower_case = True

    bert_model = 'bert-base-uncased'
    character_bert_path = './pretrained-models/general_character_bert/'

    rnn_dim = 128

    task = 'sequence_labelling'
    embedding = 'general-character-bert'

    train_batch_size = 12
    eval_batch_size = 12

    gradient_accumulation_steps = 1
    num_train_epochs = 20

    validation_ratio = 0.5
    learning_rate = 5e-5
    weight_decay = 0.1
    warmup_ratio = 0.1
    adam_epsilon = 1e-8
    max_grad_norm = 1.0
    dp_train = True
    seed = 881245421

    device = 'gpu'
```

Рисунок 3.5 – Код класу Config

Для передбачення за допомогою мережі було розроблено функцію predict_sentence. Він отримує на вхід речення, проводить всі необхідні операції над даними, передає їх на вхід до мережі та обробляє вихідний результат. На виході методу маємо класи для кожного слова у реченні.

На першому етапі роботи функції відбувається ініціалізація об’єктів, що відповідають за токенізацію (див. рисунок 3.6).

```
def predict_sentence(sentence):
    bert_tokenizer = BertTokenizer.from_pretrained(config.bert_model,
                                    do_lower_case=config.do_lower_case)

    basic_tokenizer = bert_tokenizer.basic_tokenizer
    indexer = CharacterIndexer()
```

Рисунок 3.6 – Код ініціалізації об’єктів, що відповідають за токенізацію

Наступний етап – токенізація. На цьому етапі за допомогою класу BertTokenizer відбувається токенізація речення. Далі до токенізованого речення додаються токени «[CLS]» та «[SEP]», які є необхідними для роботи моделі мови CharacterBERT. Код токенізації зображено на рисунку 3.7.

```
tokens = basic_tokenizer.tokenize(sentence)
tokens = ['[CLS]', *tokens, '[SEP]']
batch = [tokens]
```

Рисунок 3.7 – Код токенізації

```
n = len(ids[0])
ids = indexer.as_padded_tensor(batch)
attention_mask = [[1] * n]
segment_ids = [[0] * n]
```

Рисунок 3.8 – Код формування вхідних даних

Далі формуються масиви вхідних даних. На цьому етапі масив токенів формується у двомірний масив індексів символів слів. Додатково створюються маска уваги (attention_mask) та масив сегментів (segment_ids). Код формування вхідних даних зображено на рисунку 3.8.

Для того, щоб передати масив з даними до моделі у бібліотеці PyTorch їх необхідно помістити в об’єкт класу Tensor.

Отримані тензори необхідно присвоїти до пристрою, на якому виконується мережа. Далі, щоб отримати результати передбачення, необхідно викликати метод «predict» класу моделі. Код отримання результатів передбачення зображено на рисунку 3.10.

```
mask_tensor.to(config.device)
segment_tensor.to(config.device)
ids.to(config.device)

res = model.predict(ids, segment_tensor, mask_tensor)
```

Рисунок 3.9 – Код отримання результатів передбачення

Далі отриманий результат обробляється та повертається з функції (див. рисунок 3.10)

```
label_ids = res[0]
out_labels = out_labels[1:len(out_labels) - 1]
tokens = tokens[1:len(tokens) - 1]

return tokens, out_labels
```

Рисунок 3.10 – Код обробки результатів передбачення

У результаті виконання функції predict_sentence отримаємо масив токенів та відповідний масив класів сутностей.

3.3 Розробка модуля обробки даних

Наступним етапом розробки додатку для розпізнавання іменованих сутностей у тексті є створення класів та функцій для обробки даних.

Процеси обробки даних для навчання та виконання відрізняються один від одного, адже при навчанні першим ділом необхідно завантажити набір даних для навчання. У якості набору даних, як правило, для задач пошуку іменованих сутностей використовують набір даних ConLL2003. Він включає у себе такі класи сутностей: імена людей, організацій, локацій та загальні імена (фільми, книги, картини, тощо). Для завантаження набору даних було розроблено функцію load_sequence_labelling_dataset. Спочатку метод визначає директорію, в якій знаходяться файли з даними для навчання (див. рисунок 3.11).

```
path = os.path.join(DATA_PATH, 'sequence_labelling', f'{step}.txt')

if data_dir is not None:
    path = os.path.join(data_dir, f'{step}.txt')
```

Рисунок 3.11 – Початок функції методу load_sequence_labelling_dataset

Наступний крок – ініціалізація основних змінних та читання файлу за допомогою функції open, що входить до стандартної бібліотеки мови програмування Python (рисунок 3.12).

```
i = 0
examples = []
with open(path, 'r', encoding='utf-8') as data_file:
    lines = data_file.readlines()
    token_sequence = []
    label_sequence = []
```

Рисунок 3.12 – Ініціалізація основних змінних та читання файлу

В основному циклі відбувається ітерація по стрічкам у файлі й формування набору даних. Код циклу зображено на рисунку 3.13.

```
for line in tqdm(lines, desc=f'reading `{os.path.basename(path)}`...'):
    splitline = line.strip().split()
    if splitline:
        token, label = splitline
        token_sequence.append(token)
        label_sequence.append(label)
    else:
        examples.append(
            SequenceLabellingExample(
                id=i,
                token_sequence=token_sequence,
                label_sequence=label_sequence,
            )
        )
        i += 1
        token_sequence = []
        label_sequence = []
```

Рисунок 3.13 – Основний цикл функції load_sequence_labelling_dataset

Далі необхідно перевести токенізацію отриманих наборів даних. Для цього було розроблено наступні функції: retokenize та build_features.

Одразу ж після завантаження набору даних викликається функція retokenize (див. рисунок 3.14).

```
data = {}
for split in ['train', 'test']:
    func = load_sequence_labelling_dataset

    data[split] = func(step=split, do_lower_case=config.do_lower_case,
data_dir=config.path_to_data)
    data[split] = data[split]

    retokenize(data[split], tokenization_function)
```

Рисунок 3.14 – Код обробки набору даних

Функція retokenize токенізує дані таким чином, щоб їх можна було використовувати у моделі мови CharacterBERT. Дана функція ітерує по реченнях у наборі даних та токенізує їх.

Спочатку у циклі об’являються масиви токенів речення та класів (див. рисунок 3.15).

```
def retokenize(examples, tokenization_function):
    for i, example in tqdm.tqdm(enumerate(examples), desc='retokenizing
examples...'):
        tokens = example.token_sequence
        labels = example.label_sequence
        assert tokens
        assert len(tokens) == len(labels)
        new_tokens, new_labels = [], []
```

Рисунок 3.15 – Ініціалізація основних змінних у функції retokenize

Далі за допомогою функції токенайзера моделі мови BERT виконується токенізація кожного токена окремо (див. рисунок 3.16).

```
for token, label in zip(tokens, labels):
    retokenized_token = tokenization_function(token)
```

Рисунок 3.16 – Токенізація за допомогою BertTokenizer

У процесі токенізації первинний токен (частина мови) може бути розбитий на більш дрібні складові. Наприклад слово «don’t» буде розбито на наступні токени: «don», «’» і «t». Через це кількість окремих частин мови, яка буде подаватися у мережу на аналіз збільшується. Разом із цим необхідно пам’ятати, що кожному слову в наборі даних присвоєно певний клас сутності. Таким чином, після токенізації необхідно правильно присвоїти всі класи новим токенам. Код, що відповідає за це зображено на рисунку 3.17.

```
if retokenized_token != [token]:
    if label != 'O':
        label_pos = label[:2]
        label_type = label.split('-')[-1]
        if label_pos == 'B-':
            new_label = [label] + (len(retokenized_token) - 1) *
                        ['I-' + label_type]
        elif label_pos == 'I-':
            new_label = [label] * len(retokenized_token)
    else:
        new_label = [label] * len(retokenized_token)
    new_tokens.extend(retokenized_token)
    new_labels.extend(new_label)
else:
    new_tokens.append(token)
    new_labels.append(label)
```

Рисунок 3.17 – Код присвоєння класів після токенізації

Фінальний етап – заміна оригінального речення на токенізоване. Код зображено на рисунку 3.18.

```
if new_tokens:
    example = example._replace(token_sequence=new_tokens)
    example = example._replace(label_sequence=new_labels)
else:
    example = example._replace(token_sequence=[''])
    example = example._replace(label_sequence=['O'])

examples[i] = example
```

Рисунок 3.18 – Код заміни оригінального речення на токенізоване

У результаті виконання функції буде отримано токенізований набір даних, у якому правильно присвоєні класи сутностей.

Для перетворення токенізованих даних у вхідні дані до мережі використовується функція build_features. Перший крок виконання методу – конвертація набору даних (див. рисунок 3.19).

```
def build_features(
        args, split, tokenizer, examples, labels,
        pad_token_id, pad_token_label_id, max_seq_length):

    logging.info("Building features from data...")
    func = convert_examples_to_features_tagging
    features = func(
        args, tokenizer, examples, labels,
        pad_token_id, pad_token_label_id, max_seq_length
    )
```

Рисунок 3.19 – Виклик функції конвертації набору даних

Наступний крок – формування об’єкту TensorDataset, що буде використовуватися у процесі навчання нейронної мережі. Даний об’єкт міститиме у собі всю необхідну інформацію для навчання мережі: порядкові номера символів (input_ids), маски уваги (input_mask), масиви сегментів (segment_ids) та правильні номера класів сутностей (label_ids). Код формування TensorDataset зображено на рисунку 3.20.

```
    all_input_ids = torch.tensor([f.input_ids.tolist() for f in features],
dtype=torch.long)
    all_input_mask = torch.tensor([f.input_mask for f in features],
dtype=torch.long)
    all_segment_ids = torch.tensor([f.segment_ids for f in features],
dtype=torch.long)
    all_label_ids = torch.tensor([f.label_ids for f in features], dtype=torch.long)
    dataset = TensorDataset(all_input_ids, all_input_mask, all_segment_ids,
all_label_ids)
    return dataset
```

Рисунок 3.20 – Код формування TensorDataset

Функція convert_examples_to_features_tagging використовується для перетворення токенів у масиви порядкових номерів символів та класів у їх порядкові номери. Також у функції створюється масивив сегментів та маски уваги.

Першим етапом роботи функції є ініціалізація основних змінних, код якої зображено на рисунку 3.21.

```
def convert_examples_to_features__tagging(
        args, tokenizer, examples, labels,
        pad_token_id, pad_token_label_id, max_seq_length):
    """Converts tagging examples into pytorch tensors."""

    InputFeatures = namedtuple(
        'SequenceLabelligFeatures',
        ['input_ids', 'input_mask', 'segment_ids', 'label_ids'])

    label_map = {label: i for i, label in enumerate(labels)}

    data_iterator = tqdm.tqdm(enumerate(examples), total=len(examples))

    features = []
```

Рисунок 3.21 – Ініціалізація основних змінних у функції convert_examples_to_features_tagging

Далі необхідно проітерувати (див. рисуноку 3.22) набір даних. Це робиться за допомогою використання змінної data_iterator, що була об’явлена раніше. На кожній ітерацій токени та класи присваюються у відповідні змінні.

```
for i, example in data_iterator:
    tokens = example.token_sequence
    labels = example.label_sequence
```

Рисунок 3.22 – Початок циклу ітерації набору даних у функції convert_examples_to_features_tagging

Наступний етап – формування масиву індексів класів. Для цього використовується змінна label_map, яка є словником, ключем якого є клас, а значенням – його порядковий номер. Код формування масиву індексів класів зображено на рисунку 3.23.

```
label_ids = []
for token, label in zip(tokens, labels):
    if token.startswith('##'):
        label_ids.append(pad_token_label_id)
    else:
        label_ids.append(label_map[label])
```

Рисунок 3.23 – Код формування масиву індексів класів

Наступний етап – додавання спеціальних токенів, що використовуються у моделях мови на базі BERT (див. рисунок 3.24).

```
tokens += ["[SEP]"]
label_ids += [pad_token_label_id]
segment_ids = [0] * len(tokens)

tokens = ["[CLS]"] + tokens
label_ids = [pad_token_label_id] + label_ids
segment_ids = [0] + segment_ids
```

Рисунок 3.24 – Код додавання спеціальних токенів

Окрім додавання токенів, на цьому етапі також формується масив сегментів та для нових токенів присвоюються порядкові номери «пустих» класів. Це необхідно для того, щоб класифікатор ігнорував ці токени при аналізі речення.

Наступний етап – перетворення токенів у масиви символів та вирівнювання довжин усіх вхідних тензорів (див. рисунок 3.25).

```
input_ids = tokenizer.as_padded_tensor([tokens], maxlen=max_seq_length)[0]
input_mask = [1] * len(tokens)
padding_length = max_seq_length - len(input_mask)
input_mask += [0] * padding_length
segment_ids += [0] * padding_length
label_ids += [pad_token_label_id] * padding_length
```

Рисунок 3.25 – Код перетворення токенів у масиви символів та вирівнювання довжин усіх вхідних тензорів

Для перевірки правильності обробки даних у методі присутнє логування інформації про перших 3 речення з набору даних (див. рисунок 3.26).

```
if i < 3:
   logging.info("*** Example ***")
   logging.info("tokens: %s", " ".join([str(x) for x in tokens]))
   logging.info("input_ids: %s", " ".join([str(x) for x in input_ids]))
   logging.info("input_mask: %s", " ".join([str(x) for x in input_mask]))
   logging.info("segment_ids: %s", " ".join([str(x) for x in segment_ids]))
   logging.info("labels: %s", " ".join([str(x) for x in labels]))
   logging.info("label_ids: %s", " ".join([str(x) for x in label_ids]))
```

Рисунок 3.26 – Код логування

Останній етап – додавання результатів у масив features (див. рисунок 3.27)

```
features.append(
    InputFeatures(
        input_ids=input_ids,
        input_mask=input_mask,
        segment_ids=segment_ids,
        label_ids=label_ids)
)
```

Рисунок 3.27 – Код додавання результатів у масив features

У результаті роботи усіх описаних функцій обробки даних буде отримано об’єкт TensorDataset, який містить у собі всі необхідні вхідні дані для навчання нейронної мережі.

3.4 Розробка модуля навчання нейронної мережі

Після розробки нейронної мережі та модулів обробки даних необхідно розробити функції, що відповідають за навчання нейронної мережі.

Першим етапом навчання мережі є ініціалізація всіх необхідних об’єктів, що працюють з навчальними даними (див. рисунок 3.28). Для цього у бібліотеці PyTorch використовуються класи RandomSampler та DataLoader.

Клас DataLoader відповідає за вибір даних з набору у процесі навчання. Для того, щоб дані вибиралися у випадковому порядку необхідно ініціалізувати екземпляр класу RandomSampler та передати його в конструктор об’єкту DataLoader.

```
def train(args, dataset, model, tokenizer, labels, pad_token_label_id):

    train_dataset = dataset['train']
    train_sampler = RandomSampler(train_dataset)
    train_dataloader = DataLoader(
        train_dataset,
        sampler=train_sampler,
        batch_size=args.train_batch_size)
```

Рисунок 3.28 – Ініціалізація DataLoader

Далі необхідно задати параметри навання мережі (див. рисунок 3.29). Далі ці параметри необхідно передати до класу, що реалізує функціональність оптимізатора.

У якості оптимізатора для нейронної мережі було обрано оптимізатор Adam. У бібліотеці PyTorch для використання оптимізатору Adam необхідно створити екземпляр класу AdamW (див. рисунок 3.30). Для перевірки додано логування параметрів мережі (див. рисунок 3.31).

```
    n_train_steps__single_epoch = len(train_dataloader) //
args.gradient_accumulation_steps
    n_train_steps = n_train_steps__single_epoch * args.num_train_epochs
    args.logging_steps = n_train_steps__single_epoch

    no_decay = ["bias", "LayerNorm.weight"]
    optimizer_grouped_parameters = [
        {
            "params": [p for n, p in model.named_parameters()
                       if not any(nd in n for nd in no_decay)],
            "weight_decay": args.weight_decay,
        },
        {
            "params": [p for n, p in model.named_parameters()
                       if any(nd in n for nd in no_decay)],
            "weight_decay": 0.0
        },
    ]
```

Рисунок 3.29 – Ініціалізація параметрів навчання мережі

```
    optimizer = AdamW(optimizer_grouped_parameters, lr=args.learning_rate,
eps=args.adam_epsilon)
    scheduler = get_linear_schedule_with_warmup(
        optimizer,
        num_warmup_steps=int(args.warmup_ratio*n_train_steps),
        num_training_steps=n_train_steps
    )
```

Рисунок 3.30 – Ініціалізація оптимізатору Adam

```
        logging.info("***** Running training *****")
        logging.info("  Num examples = %d", len(train_dataset))
        logging.info("  Num Epochs = %d", args.num_train_epochs)
        logging.info(
            "  Total train batch size (w. accumulation) = %d",
            args.train_batch_size * args.gradient_accumulation_steps
        )
        logging.info("  Gradient Accumulation steps = %d",
    args.gradient_accumulation_steps)
        logging.info("  Total optimization steps = %d", n_train_steps)
        logging.info("  Using linear warmup (ratio=%s)", args.warmup_ratio)
        logging.info("  Using weight decay (value=%s)", args.weight_decay)
```

Рисунок 3.31 – Ініціалізація логування параметрів

Наступний крок – ініціалізація змінних, що будуть використовуватися у процесі навчання нейронної мережі. Ці змінні необхідні для того, щоб запам’ятовувати результати тренування мережі. Наприклад, змінна best_metric зберігає найкращу оцінку надійності мережі, що дозволяє відслідковувати прогрес навчання. Код зображено на рисунку 3.32.

```
global_step = 0
epochs_trained = 0
steps_trained_in_current_epoch = 0

tr_loss, logging_loss = 0.0, 0.0
best_metric, best_epoch = -1.0, -1
```

Рисунок 3.32 – Ініціалізація змінних, що будуть використовуватися у

Процес навчання нейронної мережі являє собой прохід по набору даних кілька разів. Повний прохід прийнято називати епохою. У кожній епосі необхідно проітерувати набір даних. На кожному етапі ітерації з набору вибирається партія тренувальних даних.

Код циклів, що відповідають за ітерацію по епохам та по даним в наборі зображено на рисунку 3.33.

```
    model.zero_grad()
    train_iterator = tqdm.trange(epochs_trained, int(args.num_train_epochs),
desc="Epoch")

    set_seed(seed_value=args.seed)
    for num_epoch in train_iterator:
        epoch_iterator = tqdm.tqdm(train_dataloader, desc="Iteration")
        for step, batch in enumerate(epoch_iterator):

            if steps_trained_in_current_epoch > 0:
                steps_trained_in_current_epoch -= 1
                continue
```

Рисунок 3.33 – Код основних циклів навчання мережі

На кожному кроці в середині епохи спочатку потрібно перевести об’єкт моделі нейронної мережі у режим навчання. Це робиться за рахунок виклику методу «train». Далі, через особливість роботи бібліотеки PyTorch, усі тензори в потоній партії необхідно прив’язати до пристрою, на якому виконується мережа. Це робиться викликом функції «to», аргументом якої є назва пристрою.

Код підготовки моделі та даних до навчання зображено на рисунку 3.34.

```
model.train()
batch = tuple(t.to(args.device) for t in batch)
```

Рисунок 3.34 – Код підготовки моделі та даних до навчання

Далі всі дані з партії необхідно передати до мережі на обробку. Для цього було сформовано об’єкт inputs, що включає в себе наступні поля: input_ids, attention_mask, token_type_ids, labels. Сформований об’єкт inputs подається на вхід до мережі. Результатом такого виконання є значення функції втрат. Це значення записується у змінну loss. Код описаного процесу зображено на рисунку 3.35.

```
inputs = {
    "input_ids": batch[0],
    "attention_mask": batch[1],
    "token_type_ids": batch[2],
    "labels": batch[3]}

outputs = model(**inputs)
loss = outputs
```

Рисунок 3.35 – Код отримання значення функції втрат

Пілся отримання значення функції втрат необхідно виконати процес навчання мережі. Код зображено на рисунку 3.36.

```
if args.gradient_accumulation_steps > 1:
    loss = loss / args.gradient_accumulation_steps

loss.backward()

tr_loss += loss.item()
if (step + 1) % args.gradient_accumulation_steps == 0:
    torch.nn.utils.clip_grad_norm_(model.parameters(),
                                   args.max_grad_norm)
    optimizer.step()
    scheduler.step()
    model.zero_grad()
    global_step += 1
```

Рисунок 3.36 – Код навчання мережі

Раз у декілька ітерацій, як правило, виконують перевірку мережі на наборі перевірки (validation). Це необхідно для того, щоб у процесі навчання мережі можна було відслідковувати те, як мережа виконується на даних, відмінних від навчальних. Код виклику функції перевірки зображено на рисунку 3.37.

```
if global_step % args.logging_steps == 0:

    results, _ = evaluate(
        args=args,
        eval_dataset=dataset["validation"],
        model=model, labels=labels,
        pad_token_label_id=pad_token_label_id
    )
```

Рисунок 3.37 – Код виклику функції перевірки мережі

За результатами перевірки модель та її внутрішні параметри зберігаються у файл для того, щоб у подальшому її можна було використовувати для передбачень. Оскільки надійність мережі іноді може бути нижчою, ніж на попередній ітерації, то перед збереженням робиться перевірка на те, чи є надійність вищою, за найкращий результат у минулому. Код збереження стану мережі зображено на рисунку 3.38.

```
metric = results['f1']

if metric > best_metric:
    best_metric = metric
    best_epoch = num_epoch

    if not os.path.exists(args.output_dir):
        os.makedirs(args.output_dir)
    model.save_pretrained(args.output_dir)
    if 'character' not in args.embedding:
        tokenizer.save_pretrained(args.output_dir)
    torch.save(args, os.path.join(args.output_dir,
                                  "training_args.bin"))
    logging.info("Saving model checkpoint to %s",
                                   args.output_dir)
```

Рисунок 3.38 – Код збереження мережі у файл

У процесі навчання, як правило, проводять перевірку мережі. Для цього використовують додатковий набір даних, який не містить даних із набору для навчання. У процесі перевірки мережа виконується на цьому наборі даних. Отримані результати порівнюються з правильними відповідями. На основі такого порівняння формуються основні метрики мережі: precision, recall та f1 score.

Для перевірки мережі було розроблено функцію evaluate. На першому етапі виконання функції створюються екземпляри класів SequentialSampler та DataLoader (див. рисунок 3.39). SequentialSampler використовується тому, що на відміну від процесу навчання при перевірці відсутня необхідність у тому, щоб подавати дані з набору у випадковому порядку.

```
def evaluate(args, eval_dataset, model, labels, pad_token_label_id):
    eval_sampler = SequentialSampler(eval_dataset)
    eval_dataloader = DataLoader(
        eval_dataset,
        sampler=eval_sampler,
        batch_size=args.eval_batch_size)
```

Рисунок 3.39 – Код створення об’єкту DataLoader фукнції evaluate

Далі потрібно ініціалізувати змінні, що будуть використовуватися у процесі перевірки мережі. Код зображено на рисунку 3.40.

```
eval_loss = 0.0
nb_eval_steps = 0
preds = None
out_label_ids = None
```

Рисунок 3.40 – Код ініціалізації основних змінних у функції evaluate

Процес перевірки являє собою прохід по набору даних для перевірки та отримання результатів передбачення мережі для кожної партії. Перед початком перевірки модель необхідно перевести у режим «виконання». Це робиться за рахунок виклику методу «eval». Код циклу зображено на рисунку 3.41.

```
model.eval()
for batch in tqdm.tqdm(eval_dataloader, desc="Evaluating"):
    batch = tuple(t.to(args.device) for t in batch)
```

Рисунок 3.41 – Код циклу функції evaluate

Для передбачення за допомогою мережі формується об’єкт input, що містить наступні поля: inoput_ids, attention_mask, token_type_ids. Цей об’єкт поднається на вхід до мережі через виклик методу «predict» об’єкту моделі. На виході отримаємо порядкові номери класів сутностей, які мережа передбачила. Код отримання результатів передбачення мережі зображено на рисунку 3.42.

```
with torch.no_grad():
    input = {
        "input_ids": batch[0],
        "attention_mask": batch[1],
        "token_type_ids": batch[2]}

    outputs = model.predict(**input)
    logits = outputs
```

Рисунок 3.42 – Код отримання результатів передбачення у функції evaluate

Далі отримані передбачення та правильні відповіді додаються до відповідних масивів. Це необхідно для того, щоб у подальшому мати змогу розрахувати метрики мережі. Код обробки результатів передбачення зображено на рисунку 3.43.

```
try:
    nb_eval_steps += 1
    if preds is None:
        preds = np.array(logits)
        out_label_ids = inputs["labels"].detach().cpu().numpy()[]
    else:
        preds = np.append(preds, logits, axis=0)
        out_label_ids = np.append(out_label_ids,
                inputs["labels"].detach().cpu().numpy(), axis=0)
except Exception as e:
    print(e)
```

Рисунок 3.43 – Код обробки результатів передбачення у функції evaluate

Наступний крок – формування метрик мережі (див. рисунок 3.44).

```
label_map = {i: label for i, label in enumerate(labels)}
out_label_list = [[] for _ in range(out_label_ids.shape[0])]
preds_list = [[] for _ in range(out_label_ids.shape[0])]
for i in range(out_label_ids.shape[0]):
    for j in range(out_label_ids.shape[1]):
        if out_label_ids[i, j] != pad_token_label_id:
            out_label_list[i].append(label_map[out_label_ids[i][j]])
            preds_list[i].append(label_map[preds[i][j]])

results = {
    "precision": seqeval_metrics.precision_score(out_label_list, preds_list),
    "recall": seqeval_metrics.recall_score(out_label_list, preds_list),
    "f1": seqeval_metrics.f1_score(out_label_list, preds_list),
}
```

Рисунок 3.44 – Код формування метрик мережі

Пілся того, як метрики було сформовано, необхідно їх прологувати, щоб можна було відслідковувати прогрес навчання мережі. Код логування та повернення результату виконання функції зображено на рисунку 3.45.

```
logging.info("***** Eval results *****")
for key in sorted(results.keys()):
    logging.info("  %s = %s", key, str(results[key]))

return results, preds_list
```

Рисунок 3.45 – Код логування та повернення результату виконання функції evaluate

Отже, для навчання мережі було розроблено функцію train, яка для перевірки у процесі навчання викликає функцію evaluate. У результаті виконання функції train буде отримано навчену мережу.

3.5 Розробка веб-сервісу

Наступний етап – розробка веб-сервісу, який надає можливість використовувати розроблений раніше модуль пошуку іменованих сутностей.

Додаток розроблено з використанням мови програмування C# та фреймворку ASP.NET Core.

В ASP.NET Core запит після первинної обробки направляється на Controller, у залежності від шляху запиту. Тому для того, щоб обробляти запити, необхідно створити клас, що унаслідується від Controller. Код класу зображено на рисунку 3.46.

```
[ApiController]
[Route("api/v1/[controller]")]
public class NerController : ControllerBase
{

    private readonly NerPythonService nerService;

    public NerController(NerPythonService nerService)
    {
        this.nerService = nerService;
    }

    [HttpPost]
    public async Task<IActionResult> Get([FromBody] RequestDto req)
    {
        return Ok(await nerService.GetAsync(req.Text));
    }
}
```

Рисунок 3.46 – Код класу NerController

Процес комунікації між нейронною мережею та веб-сервісом здійснюється шляхом локальних HTTP запитів. Отриманий запит відсилається локальний веб-сервер, який написаний мовою програмування Python. Дане рішення обумовлене тим, що процес ініціалізації мережі є досить затратним. Сервер на Python дозволяє ініціалізувати мережу на старті та використовувати її повторно.

Локальний веб-сервер для нейронної мережі написаний з використанням бібліотеки Flask. Код веб-серверу зображено на рисунку 3.47.

```python
app = flask.Flask("ner_python_api")

@app.route('/', methods=['POST'])
def home():
    body = request.json

    if 'text' in body:
        text = body['text']
    elif 'Text' in body:
        text = body['Text']
    else:
        return 'Error: No text provided!'

    return jsonify(predict.run(text))

app.run(port=11155)
```

Рисунок 3.47 – Код локального веб-серверу для нейронної мережі

Оскільки C# це строго типізована мова програмування, то важливим є створення класів усіх моделей, що будуть використовуватися у додатку.

Для відправки запиту на веб-сервер мережі було створено клас RequestDto, код якого зображено на рисунку 3.48.

```
public class RequestDto
{
    public string Text { get; set; }
}
```

Рисунок 3.48 – Код класу RequestDto

Результат з веб-серверу нейронної мережі приходить у вигляді масиву об’єктів, що містять наступні поля: оригінальне речення, масив токенів та масив класів, що передбачила мережа. Для опису цього об’єкту було створено клас NerResultDto, код якого зображено на рисунку 3.49.

```
public class NerResultDto
{
    public string Sentence { get; set; }
    public ICollection<string> Tokens { get; set; }
    public ICollection<string> Labels { get; set; }
}
```

Рисунок 3.49 – Код класу NerResultDto

Для опису сутності було створено клас NerEntity, код якого зображено на рисунку 3.50.

```
public class NerEntity
{
    public string Label { get; set; }
    public string Tokens { get; set; }
}
```

Рисунок 3.50 – Код класу NerEntity

Для формування відповіді на запит створено клас ResultDto, що містить у собі оригінальне речення та масив об’єктів NerEntity. Код класу зображено на рисунку 3.51.

```
public class ResultDto
{
    public string Senctence { get; set; }
    public ICollection<NerEntity> Enitites { get; set; }
}
```

Рисунок 3.51 – Код класу ResultDto

Для комунікації з веб-сервісом нейронної мережі та обробки результатів передбачення було створено клас NerPythonService.

Для надсилання веб-запитів використовується екземпляр класу HttpClient, а для читання файлу конфігурації об’єкт, що реалізує інтерфейс IConfiguration. Код ініціалізації HttpClient та IConfiguration, а також їх присвоєння у конструкторі класу NerPythonService зображено на рисунку 3.52.

```
private readonly HttpClient httpClient;
private readonly IConfiguration configuration;

public NerPythonService(HttpClient httpClient,
                        IConfiguration configuration)
{
    this.httpClient = httpClient;
    this.configuration = configuration;
}
```

Рисунок 3.52 – Змінні та конструктор класу NerPythonService

Відправка запиту на локальний веб-сервіс відбувається у методі «GetAsync».

Спочатку формується тіло запиту. Для цього створюється об’єкт типу RequestDto, що містить текст, у якому необхідно знайти сутності. Далі цей об’єкт необхідно сереалізувати та створити екземпляр класу StringContent. Створення тіла запиту зображено на рисунку 3.53.

```
var jsonConfig = new JsonSerializerOptions
{
    PropertyNameCaseInsensitive = true
};

var reqContent = new StringContent(
    JsonSerializer.Serialize(new RequestDto {Text = text}),
    Encoding.UTF8,
    "application/json");
```

Рисунок 3.53 – Створення тіла запиту у методі GetAsync

Для відправки запиту використовується метод «PostAsync» класу HttpClient. Перед відправкою запиту необхідно отримати адресу локального веб-серверу. Код відправки запиту зображено на рисунку 3.54.

```
var url = configuration["NER:PythonEndpoint"];
var response = await httpClient.PostAsync(url, reqContent);
```

Рисунок 3.54 – Код відправки запиту в методі GetAsync

Результат запиту необхідно десеріалізувати та передати на обробку в метод ParseResponse (див. рисунок 3.55).

```
if (response is null || !response.IsSuccessStatusCode)
    throw new Exception("Server error!");

var content = await response.Content.ReadAsStringAsync();
var results = JsonSerializer
    .Deserialize<IEnumerable<NerResultDto>>(content, jsonConfig);

return ParseResponse(results);
```

Рисунок 3.55 – Код обробки запиту в методі GetAsync

Метод ParseResponse відповідає за перетворення даних у більш зручний формат. У цьому методі з набору токенів та їх класів формується масив об’єктів NerEntity. Кожен такий об’єкт описує окрему сутність.

```
var resultDto = new ResultDto { Senctence = result.Sentence };

var labels = result.Labels;
var tokens = result.Tokens;

var label = string.Empty;
var entTokens = new StringBuilder();

var entites = new List<NerEntity>();
```

Рисунок 3.56 – Ініціалізація основних змінних у методі ParseResponse

У методі в циклі виконується прохід по всім об’єктам NerResultDto, що були отримані у відповідь з веб-сервісу нейронної мережі. Першим етапом у циклі є ініціалізація необхідних змінних (див. рис. 3.56).

Далі виконується прохід по всіх парах (токен, клас). В залежності від того, який тип класу («B» – початок класу, «I» – продовження), або створюється нова сутність NerEntity, або до поточної сутності додаються нові токени. Код циклу зображено на рисунку 3.57.

```
foreach (var pair in tokens.Zip(labels))
{
    if (pair.Second.StartsWith("B"))
    {
        if (!string.IsNullOrEmpty(label))
            entites.Add(new NerEntity
            {
                Label = label,
                Tokens = entTokens.ToString()
            });

        label = pair.Second.Split('-').Last();
        entTokens = new StringBuilder();
        entTokens.Append(pair.First);
    }
    else if (pair.Second.StartsWith("I") &&
             pair.Second.Split('-').Last() == label)
        entTokens.Append($" {pair.First}");
}
```

Рисунок 3.57 – Код циклу прохода по парах (токен, клас)

Таким чином, розроблені класи NerController та NerPythonService дозволяють приймати та обробляти запити на пошук іменованих сутностей у тексті. Клас NerPythonService, у свою чергу, за допомогою використання HTTP запитів здійснює комунікацію з локальним веб-сервісом нейронної мережі для отримання результатів передбачення.

3.6 Висновки

У третьому розділі було обґрунтовано вибір мов програмування та технологій, які будуть використовуватися при розробці програмного продукту та наведено основні їх переваги.

У результаті аналізу було обрано мову програмування C# та технологію ASP.NET Core для розробки веб-сервісу.

Для розробки модуля розпізнавання іменованих сутностей у тексті було обрано мову програмування Python та бібліотеку PyTorch для створення нейронної мережі.

Було проведено розробку нейронної мережі, модулів її навчання та обробки даних.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Тестування системи

Одним із важливих етапів життєвого циклу будь-якого програмного продукту є тестування. Воно необхідне для того, щоб виявити дефекти у системі.

Оскільки система є веб-сервісом, тому зв’язок із ним виконується шляхом виклику API через HTTP запити. Таким чином необхідно використовувати додаткове програмне забезпечення для надсилання HTTP запитів. Для таких цілей чудово підходить додаток Postman, який дозволяє проводити тестування будь-якого API.

Для тестування необхідно виконати декілька запитів та перевірити їх результати. Тестування першого запиту зображено на рисунку 4.1.

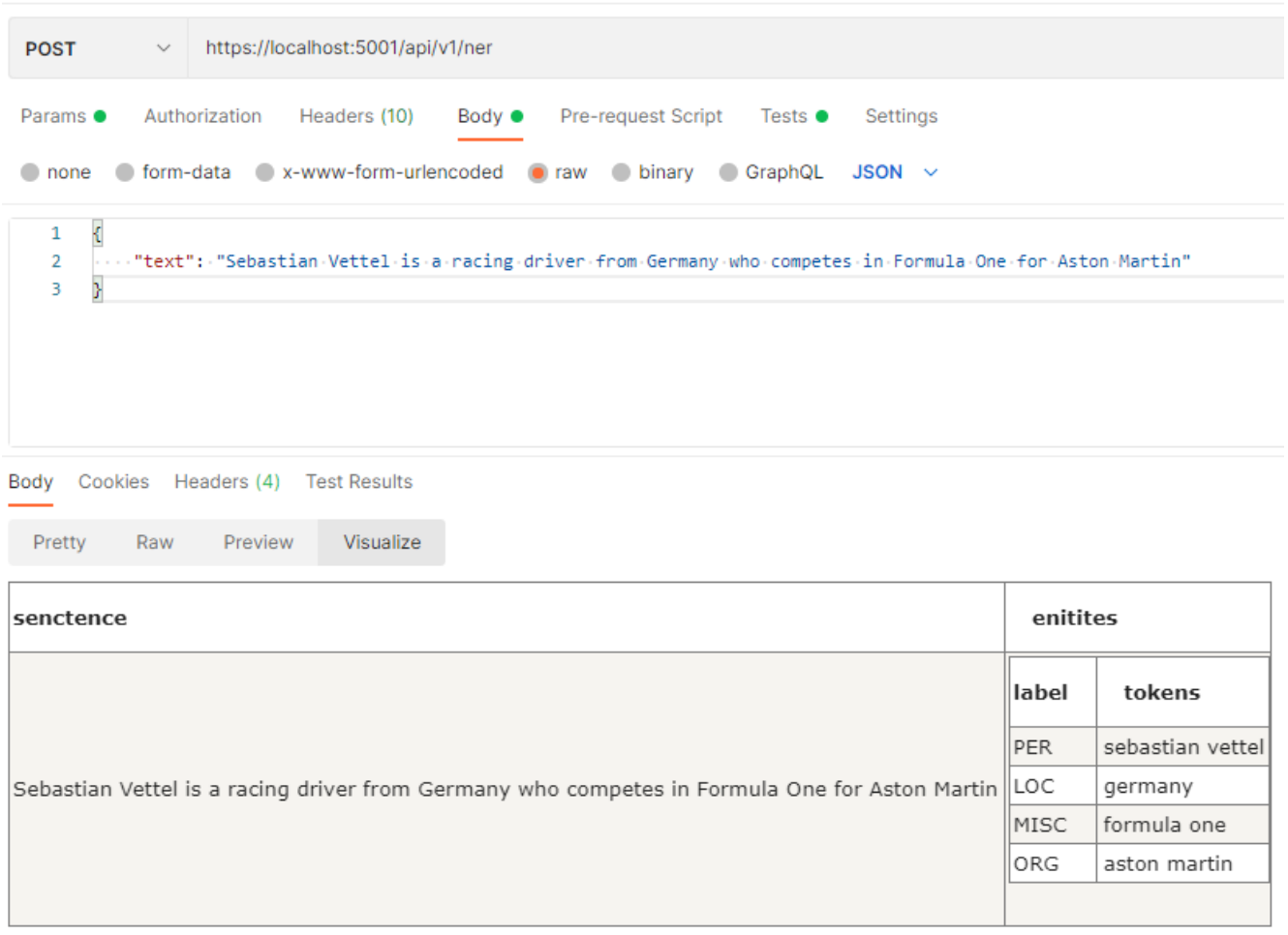

Рисунок 4.1 – Результат першого запиту

У першому випадку в реченні «Sebastian Vettel is a racing driver from Germany who competes in Formula One for Aston Martin» є чотири іменовані сутності: «Sebastian Vettel» (ім’я людини), «Germany» (назва країни, що попадає під категорію локацій), «Formula One» (назва чемпіонату, що попадає в категорію «інше») та «Aston Martin» (назва організації). Усі сутності були розпізнані системою.

Далі необхідно протестувати як система вестиме себе у випадку, коли є по кілька сутностей одного класу. Речення «Max Emilian Verstappen was born on 30 September 1997 in Hasselt, Belgium and has a younger sister, Victoria.» має дві сутності класу PER («Max Emilian Verstappen» та «Victoria») та дві сутності класу LOC («Hasselt» та «Belgium»). Результат тестування запиту зображено на рисунку 4.2.

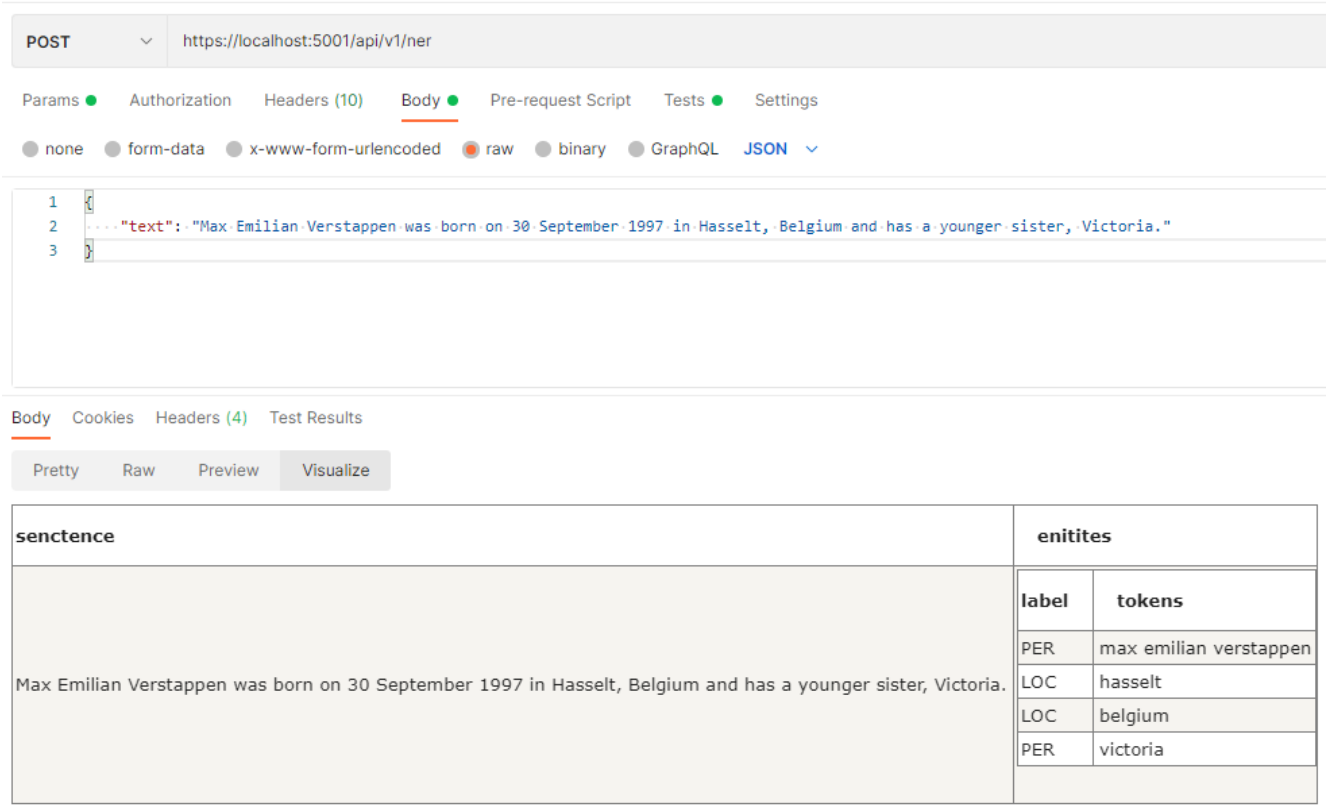

Рисунок 4.2 – Результат другого запиту

Результат тестування другого запиту є успішним. Система знайшла всі необхідні сутності.

Далі необхідно протестувати, як мережа вестиме себе у випадку, коли є більше одного речення у тексті. Результат запиту зображено на рисунку 4.3.

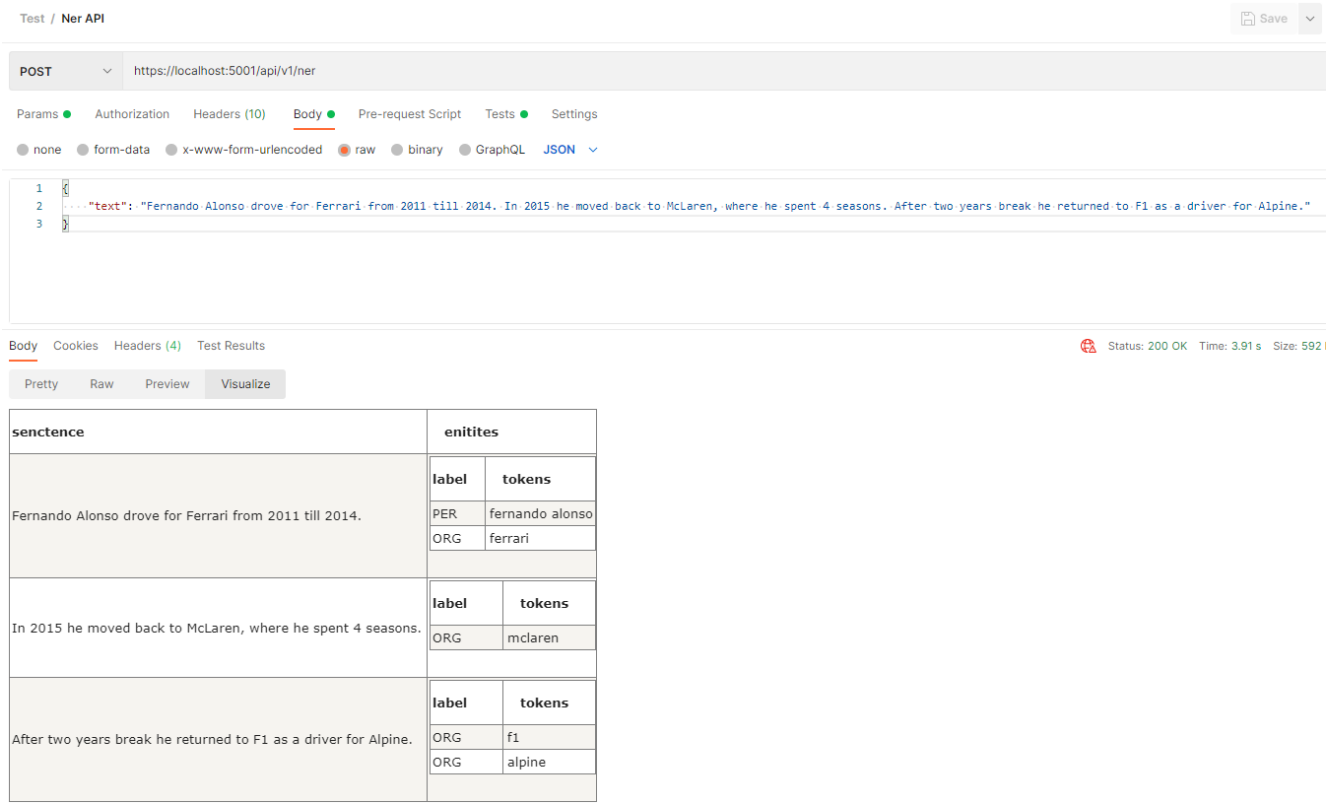

Рисунок 4.3 – Результат третього запиту

Текст запиту містив 3 речення. У результаті роботи системи, текст був правильно розділений на речення.

У першому реченні «Fernando Alonso drove for Ferrari from 2011 till 2014.» Система розпізнала сутності «Fernando Alonso» та «Ferrari».

У другому реченні «In 2015 he moved back to McLaren, where he spent 4 seasons.» Система розпізнала сутність «McLaren».

У третьому реченні «After two years break he returned to F1 as a driver for Alpine.» Система розпізнала сутності «F1» та «Alpine».

Таким чином, у третьому запиті всі сутності були розпізнані правильно.

У результаті тестування додатку було доведено, що його функціонал працює відповідно до технічного завдання.

4.2 Розробка інструкції користувача

Першим етапом розробки інструкції користувача є визначення технічних вимог програмного продукту. У таблицях 4.1 та 4.2 наведено мінімальну та рекомендовану конфігурацію персонального комп'ютера для запуску програми.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	32-розрядний (x86) або процесор з тактовою частотою 2 ГГц
Об'єм оперативної пам'яті	4 ГБ для 32-розрядної системи і 8 ГБ для 64-разрядної системи
Вільне місце на жорсткому диску	1 ГБ
Графічний прискорювач	Інтегрований графічний прискорювач, сумісний з DirectX9
Операційна система	Windows 10

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 3 ГГц
Об'єм оперативної пам'яті	4 ГБ для 32-розрядної системи і 16 ГБ для 64-разрядної системи
Вільне місце на жорсткому диску	1 ГБ
Графічний прискорювач	Графічний прискорювач, сумісний з DirectX9 та об'ємом відеопам'яті 4 ГБ
Операційна система	Windows 10

Оскільки система працює за рахунок використання декількох веб-серверів, то процес його інсталяції потребує декілька додаткових кроків.

Спочатку потрібно стартувати внутрішній сервер. Для цього в папці «ner» необхідно запустити файл «server.py». Для цього потрібно виконати команду «python server.py» у консолі команд. Додатково у цьому ж файлі можна редагувати параметри серверу, наприклад порт.

Процес запуску серверу зображено на рисунку 4.4.

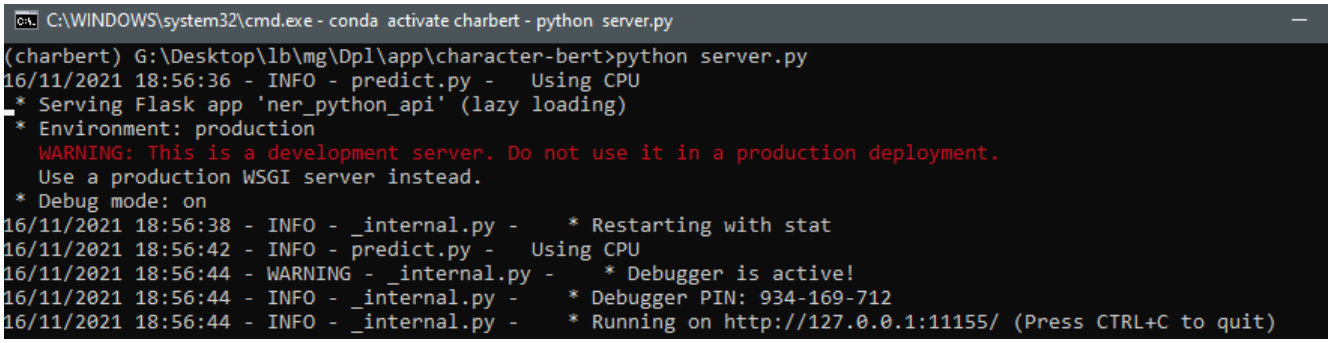

Рисунок 4.4 – Старт внутрішнього серверу

Наступний крок – налаштування середовища. Для цього необхідно у директорії «NerService» (рисунок 4.5) відредагувати файл з параметрами «appsettings.json».

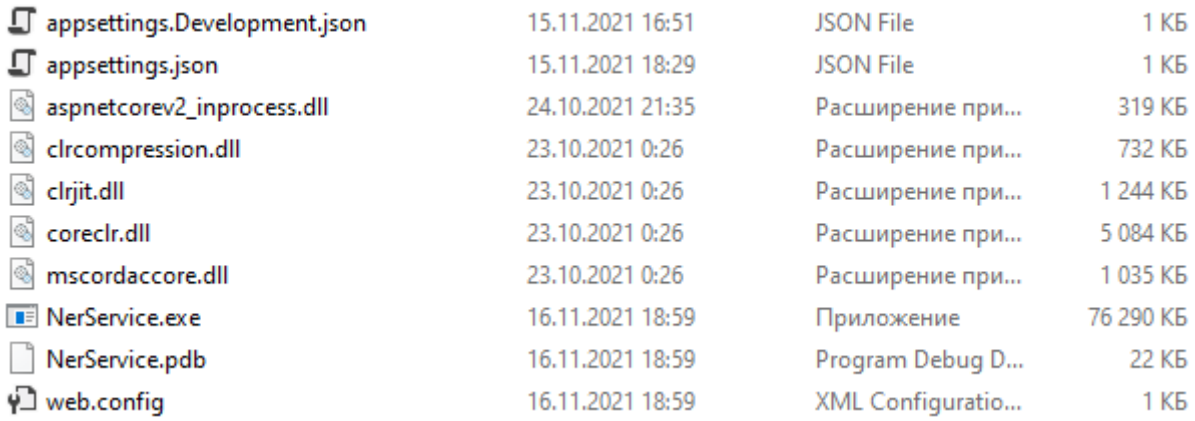

Рисунок 4.5 – Директорія «NerService»

У файлі «appsettings.json» необхідно вказати адресу внутрішнього серверу у полі «PythonEndpoint». Приклад файлу зображено на рисунку 4.6.

Рисунок 4.6 – Приклад файлу «appsettings.json»

Далі необхідно запустити файл «NerService.exe». Після того, як сервер стартував, можна надсилати HTTP запити за його адресую. Відповідь на запит міститиме набір сутностей для кожного речення у тексті.

4.3 Висновки

У четвертому розділі було проведено тестування додатку, у результаті якого було доведено його повну працездатність та відповідність поставленому технічному завданню.

Розроблено інструкцію користувача по встановленню та використанню програмного продукту.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету: Войтко Вікторія Володимирівна (к.т.н., доц. кафедри ПЗ ВНТУ), Черноволик Галина Олександрівна (к.т.н., доц. кафедри ПЗ ВНТУ), Коваленко Олена Олексіївна (к.т.н., доц. кафедри ПЗ ВНТУ). Для проведення технологічного аудиту було використано таблицю 5.1 [22] в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження табл. 5.1

5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансуванн я є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництв о та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Войтко В.В	Коваленко О.О.	Черноволик Г.О.
	Бали, виставлені експертами:		
1	4	4	4
2	2	2	2
3	3	2	3
4	3	4	3
5	2	2	2
6	4	3	3
7	3	3	3
8	4	4	4
9	2	2	1
10	3	2	3
11	3	4	4
12	4	3	3
Сума балів	$СБ_1$=37	$СБ_2$=35	$СБ_3$=35
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{37+35+35}{3} = 35.6$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 38 бали, що згідно таблиці 5.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Методи і засоби веб-системи для пошуку іменованих сутностей у тексті з

використанням нейронних мереж, що розробляються в маістерській роботі будуть цікаві підприємствам, що займаються обробкою та аналізом текстових даних (статей, новин, тощо).

Порівняємо нову розробку, що розробляється в магістерській роботі з аналогом, який існує на ринку.

В якості аналога для розробки було обрано сервіс пошуку іменованих сутностей в Microsoft Azure Cognitive Service. Основним недоліком аналога є низька надійність при роботі з текстом, що містить помилки.

У розробці дана проблема вирішується за рахунок використання моделі мови CharacterBERT, яка здатна більш точно проводити аналіз слів, що містять помилки.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом. В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 5.4 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Точність пошуку, бали	9	9	1	30%
Точність пошуку на даних з помилками, бали	5	8	1,6	30%
Середній час відповіді на запит *(менше – краще)*, мс	110	100	1,1	10%
Кількість параметрів слова при аналізі, шт	768	768	1	20%
Використання ресурсів комп’ютера, %	80	80	1	10%

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.5.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{5.1}$$

або

$$q_i = \frac{P_{Бi}}{P_{Hi}} \tag{5.2}$$

де P_{Hi}, $P_{Бi}$ – числові значення *i*-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{9}{9} = 1;$$

$$q_2 = \frac{8}{5} = 1{,}6;$$

$$q_3 = \frac{110}{100} = 1{,}1;$$

$$q_4 = \frac{768}{768} = 1;$$

$$q_5 = \frac{80}{80} = 1.$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i, \tag{5.3}$$

$$К_{я.в.} = 1 \cdot 0{,}3 + 1{,}6 \cdot 0{,}3 + 1{,}1 \cdot 0{,}1 + 1 \cdot 0{,}2 + 1 \cdot 0{,}1 = 2{,}18$$

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.4}$$

де $I_{m.n.}$ – індекс технічних параметрів; $I_{e.n.}$ – індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \quad (5.5)$$

де P_{Hei}, $P_{Бei}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$\text{I}_{\text{е.п.}} = \frac{6000}{6250} = 0{,}96;$$

$$\text{К} = \frac{2{,}18}{0{,}96} = 2{,}27.$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде конкурентоспроможніше, ніж конкурентний товар.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$\text{З}_\text{О} = \frac{\text{М}}{\text{Т}_\text{р}} * t \text{ (грн)} \quad (5.6)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

T_P – число робочих днів в місяці; приблизно $T_P \approx 21...23$ дні;

t – число робочих днів роботи дослідника.

Для розробки засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж необхідно залучити програміста з

посадовим окладом 8000 грн. Кількість робочих днів у місяці складає 22, а кількість робочих днів програміста складає 51. Зведемо сумарні розрахунки до таблиця 5.5.

Таблиця 5.5 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	12000	545,5	5	2727
Програміст	8000	363,6	51	18545
Всього				21273

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{дод}}{100\%} \quad (5.7)$$

$$З_д = 0{,}11 * 21273 = 2340 \text{ (грн)}$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.3):

$$Н_{ЗП} = (З_о + З_д) * \frac{\beta}{100} \text{ (грн)} \quad (5.8)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді**:**

$$Н_{ЗП} = (21273 + 2340) * \frac{22}{100} = 5194{,}8 \text{ (грн)}$$

4. Витрати на матеріали М та комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \text{ грн.,} \quad (5.9)$$

де $Н_i$ – витрати матеріалу *і*-го найменування, кг;

$Ц_i$ – вартість матеріалу *і*-го найменування, грн./кг.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

$В_i$ – маса відходів матеріалу *і*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *і*-го найменування, грн/кг;

n – кількість видів матеріалів.

Таблиця 5.6 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір	140	1	140
Ручка	20	1	20
CD-диск	12	1	12
Флешка	155	1	155
Всього			327
З врахуванням коефіцієнта транспортування			359,7

5. Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення необхідного для проведення дослідження. Для нової розробки використовувались безкоштовні програмні засоби.

6. Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$A = \frac{Ц \cdot T}{Ткор \cdot 12} \quad [грн], \tag{5.10}$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункту 137.3.3 Податкового кодексу амортизація нараховується на основні засоби вартістю понад 2500 грн. В нашому випадку для написання магістерської роботи використовувався персональний комп'ютер вартістю 45000 грн.

$$\text{А} = \frac{45000 \cdot 1}{2 \cdot 12} = 1875$$

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$\text{В}_\text{е} = \sum_{\text{i}=1}^{n} \frac{W_{yt} \cdot t_i \cdot \text{Ц}_\text{е} \cdot \text{К}_\text{впі}}{\eta_\text{i}} \tag{5.11}$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$B_e = \frac{0{,}3 \cdot 250 \cdot 4{,}1 \cdot 0{,}5}{0{,}8} = 192{,}19$$

Витрати на службові відрядження, витрати на роботи, які виконують сторонні підприємства, установи, організації та інші витрати в нашому досліджені не враховуються оскільки їх не було.

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \qquad (5.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$B_{нзв} = 21273 \cdot \frac{100}{100\%} = 21273 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$B = 21273+2340+5194{,}8+359{,}7+1875+192{,}19+21273=52507{,}1$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{B}{\eta}, \qquad (5.13)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт $\beta = 0{,}9$.

Звідси:

$$ЗВ = \frac{52507,1}{0,9} = 58341,57грн.$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_{1}^{n} \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N \right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100} \right) \quad (5.14)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

р – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х – ставка податку на прибуток. У 2021 році – 18%.

Припустимо, що при впроваджені результатів наукової розробки покращується якість програмного продукту для формування індивідуальних

тренувань. Припустимо, що ціна від зросте на 100 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 200 шт., протягом другого року – на 300 шт., протягом третього року на 350 шт. Реалізація продукції до впровадження розробки складала 1 шт., а її ціна до складає 6000 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [100 \cdot 1 + (6000 + 100) \cdot 200] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 208425{,}41 \text{ грн.}$$

$$\Delta П_2 = [100 \cdot 1 + (6000 + 100) \cdot (200 + 300)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 521120{,}83 \text{ грн.}$$

$$\Delta П_3 = [100 \cdot 1 + (6000 + 100) \cdot (200 + 300 + 350)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 885835{,}4 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \qquad (5.15)$$

$k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}} = 2…5$).

$$PV = 2 \cdot 58341{,}57 = 116682{,}54$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$E_{абс} = (ПП - PV) \tag{5.16}$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{5.17}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$ПП = \frac{208425{,}41}{(1+0{,}2)^1} + \frac{521120{,}83}{(1+0{,}2)^2} + \frac{885835{,}4}{(1+0{,}2)^3} = 1050597{,}89 \text{ грн.}$$

$$Е_{абс} = (1050597{,}89 - 116682{,}54\ ) = 933915{,}35 \text{грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.18}$$

$T_{ж}$ – життєвий цикл наукової розробки, роки.

$$\text{Е}_{\text{в}} = \sqrt[3]{1 + \frac{933915{,}35}{116682{,}54}} - 1 = 1{,}57 = 157\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f,$$

(5.19)

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{\min} = 0{,}18 + 0{,}05 = 0{,}23.$$

Так як $E_в > \tau_{\min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}$$

$$\text{Т}_{\text{ок}} = \frac{1}{1{,}57} = 0{,}6 \text{ роки}$$

(5.20)

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

5.5 Висновки до економічного розділу

Було проведено оцінку комерційного потенціалу методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж, який є на вище середньому рівні. При порівнянні нової розробки з

аналогом виявлено, що вона є якіснішою і конкурентоспроможнішою відносно аналога, а також краще по технічним і економічним показникам.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 52507,1 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 58341,57грн.

Вкладені інвестиції в даний проект окупляться через 6 місяців при прогнозованому прибутку 1050597,89грн. за три роки.

ВИСНОВКИ

У під час виконання магістерської кваліфікаційної роботи було розроблено методи та засоби веб-сервісу для розпізнавання іменованих сутностей у тексті. Для розробки було використано середовище програмування Visual Studio 2019 та Visual Studio Code. Робота оформлена згідно методичних вказівок [23].

Було проаналізовано стан даного питання на сьогоднішній день. Розглянуто основні аналоги, визначено їх особливості та недоліки і розроблено порівняння з власним програмним продуктом. У результаті аналізу обрано мови програмування C# та Python та технологій ASP.NET Core для розробки веб-сервісу та бібліотеку PyTorch для розробки нейронної мережі.

У процесі виконання роботи було зроблено: визначено найбільш ефективний підхід до пошуку іменованих сутностей у тексті, розроблено метод пошуку іменованих сутностей у тексті та метод обробки вхідних даних, розроблено схеми загального алгоритму роботи системи, розроблено веб-сервіс для взаємодії із модулем розпізнавання іменованих сутностей, розроблено нейронну мережу для розпізнавання іменованих сутностей у тексті, проведено тестування системи.

Удосконалено метод розпізнавання іменованих сутностей, що базується на використанні BiLSTM-CRF нейронних мереж, у якому, на відміну від існуючих, використовується модель мови CharacterBERT, чим вдається підвищити надійність роботи системи,.

Удосконалено метод підготовки даних до пошуку іменованих сутностей, у якому, на відміну від існуючих, текст розбивається на речення, що дозволяє дискретизувати вхідні дані й підвищити надійність пошуку іменованих сутностей.,

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ghosh S., Gunning D. Natural Language Processing Fundamentals. – Birmingham: Packt Publishing, 2018. 374 с.
2. Lane H., Hapke H., Howard C. Natural Language Processing in Action. – New York: Manning Publications, 2019. 544 с.
3. Gangmin Li, Yuming Li, Zhenjin Dai, Xutao Wang, Pin Ni, Xuming Bai. Named Entity Recognition Using BERT BiLSTM CRF for Chinese Electronic Health Records, 2019. [Електронний ресурс]. URL: https:// ieeexplore.ieee.org/document/8965823
4. В.В. Войтко, О.О. Коваленко, М.Ю. Позур. Розробка систем пошуку іменованих сутностей у тексті з використанням нейронних мереж / Матеріали міжнародної науково-практичної конференції молодих вчених та студентів «Молодь у світі сучасних технологій». – Херсон, 2020. [Електронний ресурс]. Режим доступу до ресурсу: http://kntu.net.ua/ukr/content/view/full/58984.
5. В.В. Войтко., О.О. Коваленко, М.Ю. Позур. Розробка нейронної мережі для пошуку іменованих сутностей з використання моделі мови BERT / Електронні інформаційні ресурси: створення, використання, доступ. Пам’яті Олексія Петровича Стахова. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 9-10 листопада 2021 р. – Суми/Вінниця: НІКО/ВНТУ, 2021. С. 49-52
6. Nadeau D., Satoshi S. A survey of named entity recognition and classification. 2007. [Електронний ресурс]. URL: https://nlp.cs.nyu.edu/ sekine/papers/li07.pdf
7. Named-entity recognition. [Електронний ресурс]. URL: https:// en.wikipedia.org/wiki/Named-entity_recognition
8. Azure Cognitive Services. [Електронний ресурс]. URL: https:// azure.microsoft.com/en-us/services/cognitive-services/

9. Google Natural Language. [Електронний ресурс]. URL: https://cloud.google.com/natural-language
10. Apache OpenNL. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Apache_OpenNLP.
11. General Architecture for Text Engineering. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/General_Architecture_for_Text_Engineering.
12. Convolutional neural network. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network.
13. Recurrent neural network. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Recurrent_neural_network.
14. Goldberg Y. Neural Network Methods in Natural Language Processing. – San Rafael: Morgan & Claypool Publishers, 2017. 309 с.
15. CharacterBERT. Word-Level Open-Vocabulary Representations From Characters. [Електронний ресурс]. URL: https://towardsdatascience.com/characterbert-reconciling-elmo-and-bert-for-word-level-open-vocabulary-representations-from-94037fe68b21.
16. Zhiheng Huang, Wei Xu, Kai Yu Bidirectional LSTM-CRF Models for Sequence Tagging 2015. [Електронний ресурс]. URL: https://arxiv.org/pdf/1508.01991.pdf
17. Long short-term memory. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Long_short-term_memory.
18. Conditional random field. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Conditional_random_field.
19. Jeffrey Richter CLR via C# (4th Edition) (Developer Reference), Microsoft Press; 2012. 896 ст.
20. ASP.NET Core 8 Pros and 3 Cons. [Електронний ресурс]. URL: https://www.ukad-group.com/blog/aspnet-core-8-pros-and-3-cons/

21. PyTorch vs TensorFlow — spotting the difference. [Електронний ресурс]. URL: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b.
22. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. 42 с
23. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Уклад / А.О. Семенов - Вінниця : ВНТУ, 2021. 60 с. СУЯ ВНТУ-03.02.02-П.001.01:21

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"_13_ "_вересня_2021 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доц. О.О. Коваленко
"____"_____2021 р.
Виконав:
_____студент гр. 1ПІ-20м М.Ю. Позур
"____"_____2021 р.

Вінниця – 2021 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж».

Галузь застосування – автоматизовані системи обробки даних.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 277 від « 24 » вересня 2021 р.

3. Мета та призначення розробки.

Метою роботи є підвищення надійності та реалістичності пошукового процесу іменованих сутностей шляхом використання нейронних мереж, що дозволяють реалізувати інтелектуальний пошук автоматизованого процесу ідентифікації іменованих сутностей.

Призначення роботи – підвищення ефективності та надійності автоматичної обробки текстової інформації.

4. Вихідні дані для проведення НДР

1. Zhiheng Huang, Wei Xu, Kai Yu Bidirectional LSTM-CRF Models for Sequence Tagging 2015. URL: https://arxiv.org/pdf/1508.01991.pdf.
2. Ghosh S., Gunning D. Natural Language Processing Fundamentals. – Birmingham: Packt Publishing, 2018. 374 с.
3. Goldberg Y. Neural Network Methods in Natural Language Processing. – San Rafael: Morgan & Claypool Publishers, 2017. 309 с.
4. Gangmin Li, Yuming Li, Zhenjin Dai, Xutao Wang, Pin Ni, Xuming Bai. Named Entity Recognition Using BERT BiLSTM CRF for Chinese Electronic Health Records, 2019. [Електронний ресурс]. URL: https://ieeexplore.ieee.org/document/8965823

5. Технічні вимоги

Вхідні дані – текст англійською мовою; вихідні дані – результат пошуку іменованих сутностей у вигляді списку знайдених сутностей.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до магістерської кваліфікаційної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії і етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз архітектур нейронних мереж та вибір найбільш доцільної для поставленої задачі	15.09.2021 – 30.09.2021
2	Розробка алгоритмів системи	01.10.2021 – 10.10.2021
3	Розробка методу обробки вхідних даних	11.10.2021 – 25.10.2021
4	Розробка методу пошуку іменованих сутностей	26.10.2021 – 15.11.2021
5	Економічна частина	16.11.2021 – 30.11.2021

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 20м

Науковий керівник: к.т.н. доц. Коваленко О.О.

Unicheck	
Оригінальність	**94,3 %**
Схожість	5,7 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж».

Автор _____ Позур Михайло Юрійович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

Додаток В – Лістинг програми

ner_model.py

```
from numpy import fabs
import torch
import torch.nn as nn
import torch.functional as F
from torchcrf import CRF

from modeling.character_bert import CharacterBertModel
from transformers.modeling_bert import BertPreTrainedModel

class BERT_BiLSTM_CRF(BertPreTrainedModel):

    def __init__(self, config, need_birnn=True, rnn_dim=128):
        super(BERT_BiLSTM_CRF, self).__init__(config)

        self.num_tags = config.num_labels
        self.bert = CharacterBertModel.from_pretrained('./pretrained-
models/general_character_bert/', )
        self.dropout = nn.Dropout(0.5)
        out_dim = config.hidden_size
        self.need_birnn = need_birnn

        if need_birnn:
            self.birnn = nn.LSTM(config.hidden_size, rnn_dim, num_layers=1,
bidirectional=True, batch_first=True)
            out_dim = rnn_dim*2

        self.hidden2tag = nn.Linear(out_dim, config.num_labels)
        self.crf = CRF(config.num_labels, batch_first=True)


    def forward(self, input_ids, labels, token_type_ids=None, attention_mask=None):
        emissions = self.tag_outputs(input_ids, token_type_ids, attention_mask)
        loss = -1*self.crf(emissions, labels, mask=attention_mask.byte())
        return loss


    def tag_outputs(self, input_ids, token_type_ids=None, attention_mask=None):

        outputs = self.bert(input_ids, token_type_ids=token_type_ids,
attention_mask=attention_mask)

        sequence_output = outputs[0]
        sequence_output = self.dropout(sequence_output)

        if self.need_birnn:
            sequence_output, _ = self.birnn(sequence_output)
```

```
        emissions = self.hidden2tag(sequence_output)

        return emissions

    def predict(self, input_ids, token_type_ids=None, attention_mask=None):
        emissions = self.tag_outputs(input_ids, token_type_ids, attention_mask)
        return self.crf.decode(emissions, attention_mask.byte())
```

predict.py

```
from dataclasses import field
import logging
import random
import torch
import os
import numpy as np
from collections import Counter
from transformers import modeling_bert
from ner_model import BERT_BiLSTM_CRF
from transformers.modeling_bert import BertConfig
from transformers import BertTokenizer
from modeling.character_bert import CharacterBertModel
from utils.character_cnn import CharacterIndexer
from config import Config, Arguments
from utils.data import retokenize, build_features
from data import load_sequence_labelling_dataset
from utils.training import train, evaluate
import hiddenlayer as hl
from torchsummary import summary
import nltk.data
import nltk

tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

def set_seed(seed_value):
    """ Sets the random seed to a given value. """
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed_value)
    logging.info("Random seed: %d", seed_value)


config = Config()

logging.basicConfig(
    format="%(asctime)s - %(levelname)s - %(filename)s -   %(message)s",
```

```
    datefmt="%d/%m/%Y %H:%M:%S",
    level=logging.INFO)


config.device = torch.device("cpu")
logging.info("Using CPU")


labels = []

with open(os.path.join(config.output_dir, 'labels.txt'), 'r', encoding='utf8') as
file:
    for line in file:
        labels.append(line)

num_labels = len(labels)

bert_config = BertConfig.from_pretrained(config.bert_model, num_labels=num_labels)

model = BERT_BiLSTM_CRF(bert_config, rnn_dim=config.rnn_dim)


pad_token_id = None
pad_token_label_id = 0

state_dict = torch.load(
    os.path.join(config.output_dir, 'pytorch_model.bin'),
map_location=config.device)

model.load_state_dict(state_dict, strict=True)

model.to(config.device)

def predict_sentence(sentence):
    bert_tokenizer = BertTokenizer.from_pretrained(config.bert_model,
do_lower_case=config.do_lower_case)

    basic_tokenizer = bert_tokenizer.basic_tokenizer
    indexer = CharacterIndexer()

    tokens = basic_tokenizer.tokenize(sentence)
    tokens = ['[CLS]', *tokens, '[SEP]']
    batch = [tokens]
    ids = indexer.as_padded_tensor(batch)

    n = len(ids[0])
    attention_mask = [[1] * n]
    segment_ids = [[0] * n]

    mask_tensor = torch.LongTensor(attention_mask)
```

```
    segment_tensor = torch.LongTensor(segment_ids)

    mask_tensor.to(config.device)
    segment_tensor.to(config.device)
    ids.to(config.device)

    print(tokens)

    res = model.predict(ids, segment_tensor, mask_tensor)

    label_ids = res[0]
    out_labels = [labels[x].replace('\n', '') for x in label_ids]

    out_labels = out_labels[1:len(out_labels) - 1]
    tokens = tokens[1:len(tokens) - 1]

    return tokens, out_labels

def run(text):
    res = []

    for sent in tokenizer.tokenize(text):
        tokens, labels = predict_sentence(sent)
        res.append({ "sentence": sent, "tokens": tokens, "labels": labels })

    return res


def visualize(model):
    args = {
        'input_ids': torch.zeros(1, 5, 50, dtype=torch.int32),
        'labels': torch.zeros(5, dtype=torch.int32),
        'token_type_ids': torch.zeros(5, dtype=torch.int8),
        'attention_mask': torch.ones(5, dtype=torch.int8)
    }

    to_in = (torch.zeros(1,5,50, dtype=torch.long), torch.zeros(1, 5,
dtype=torch.long), torch.zeros(1, 5, dtype=torch.long), torch.ones(1, 5,
dtype=torch.long))

    summary(model, [(1,5,50), (1, 5), (1, 5), (1, 5)],1, config.device,
[torch.long, torch.long, torch.long, torch.long])
```

train.py

```
from dataclasses import field
import logging
import random
import torch
import os
import numpy as np
from collections import Counter
from transformers import modeling_bert
from ner_model import BERT_BiLSTM_CRF
from transformers.modeling_bert import BertConfig
from transformers import BertTokenizer
from modeling.character_bert import CharacterBertModel
from utils.character_cnn import CharacterIndexer
from config import Config, Arguments
from utils.data import retokenize, build_features
from data import load_sequence_labelling_dataset
from torch.nn import CrossEntropyLoss
from utils.training import train, evaluate

def set_seed(seed_value):
    """ Sets the random seed to a given value. """
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed_value)
    logging.info("Random seed: %d", seed_value)


config = Config()

logging.basicConfig(
    format="%(asctime)s - %(levelname)s - %(filename)s -   %(message)s",
    datefmt="%d/%m/%Y %H:%M:%S",
    level=logging.INFO)

if torch.cuda.is_available():
    assert torch.cuda.device_count() == 1  # This script doesn't support multi-gpu
    config.device = torch.device("cuda")
    logging.info("Using GPU (`%s`)", torch.cuda.get_device_name(0))
else:
    config.device = torch.device("cpu")
    logging.info("Using CPU")

bert_tokenizer = BertTokenizer.from_pretrained(config.bert_model,
do_lower_case=config.do_lower_case)

tokenizer = bert_tokenizer.basic_tokenizer
```

```
indexer = CharacterIndexer()

tokenization_function = tokenizer.tokenize

data = {}
for split in ['train', 'test']:
    func = load_sequence_labelling_dataset

    data[split] = func(step=split, do_lower_case=config.do_lower_case,
data_dir=config.path_to_data)
    data[split] = data[split][:int(len(data[split]) * 0.2)]

    retokenize(data[split], tokenization_function)

logging.info('Splitting training data into train / validation sets...')
data['validation'] = data['train'][:int(config.validation_ratio *
len(data['train']))]
data['train'] = data['train'][int(config.validation_ratio * len(data['train'])):]
logging.info('New number of training sequences: %d', len(data['train']))
logging.info('New number of validation sequences: %d', len(data['validation']))

counter_all = Counter(
            [label
             for example in data['train'] + data['validation'] + data['test']
             for label in example.label_sequence])
counter = Counter(
            [label
             for example in data['train']
             for label in example.label_sequence])

# Maximum sequence length is either 512 or maximum token sequence length + 5
max_seq_length = min(
    512,
    5 + max(
        map(len, [
            e.token_sequence
            for e in data['train'] + data['validation'] + data['test']
        ])
    )
)

labels = sorted(counter_all.keys())
labels.append('PAD')
num_labels = len(labels)

bert_config = BertConfig.from_pretrained(config.bert_model, num_labels=num_labels)

model = BERT_BiLSTM_CRF(bert_config, rnn_dim=config.rnn_dim)
```

```
logging.info("Goal: predict the following labels")
for i, label in enumerate(labels):
    logging.info("* %s: %s (count: %s)", label, i, counter[label])


pad_token_id = None
pad_token_label_id = labels.index('PAD')

with open(os.path.join(config.output_dir, 'labels.txt'), 'w', encoding="utf-8") as
file:
    for label in labels:
        file.write(f'{label}\n')

dataset = {}

for split in data:
    dataset[split] = build_features(
        config,
        split=split,
        tokenizer=indexer,
        examples=data[split],
        labels=labels,
        pad_token_id=pad_token_id,
        pad_token_label_id=pad_token_label_id,
        max_seq_length=max_seq_length)


model.to(config.device)

global_step, train_loss, best_val_metric, best_val_epoch = train(
    args=config,
    dataset=dataset,
    model=model,
    tokenizer=tokenizer,
    labels=labels,
    pad_token_label_id=pad_token_label_id
)
logging.info("global_step = %s, average training loss = %s", global_step,
train_loss)
logging.info("Best performance: Epoch=%d, Value=%s", best_val_epoch,
best_val_metric)
```

config.py

```
from torch._C import device


class Config():
```

```
    #path_to_data = './data/wikian/'
    path_to_data = './data/conll/'
    output_dir = './results/'

    do_lower_case = True

    bert_model = 'bert-base-uncased'
    character_bert_path = './pretrained-models/general_character_bert/'

    rnn_dim = 300

    task = 'sequence_labelling'
    embedding = 'general-character-bert'

    train_batch_size = 12
    eval_batch_size = 12

    gradient_accumulation_steps = 1
    num_train_epochs = 5

    validation_ratio = 0.5
    learning_rate = 5e-5
    weight_decay = 0.1
    warmup_ratio = 0.1
    adam_epsilon = 1e-8
    max_grad_norm = 1.0
    dp_train = True
    seed = 881245421

    device = 'gpu'

class Arguments:
    task = 'sequence_labelling'
    embedding = 'general-character-bert'
```

data.py

```
import random
import logging
from collections import namedtuple

import tqdm
import numpy as np
import torch
from torch.utils.data import TensorDataset


def retokenize(examples, tokenization_function):
```

```
    for i, example in tqdm.tqdm(enumerate(examples), desc='retokenizing
examples...'):
        if type(example).__name__ == 'ClassificationExample':
            assert example.tokens_a
            if example.tokens_b is not None:
                assert example.tokens_b
            assert example.label

            new_tokens_a = []
            for token_a in example.tokens_a:
                new_tokens_a.extend(tokenization_function(token_a))
            example = example._replace(tokens_a=new_tokens_a if new_tokens_a else
['']) 

            if example.tokens_b is not None:
                new_tokens_b = []
                for token_b in example.tokens_b:
                    new_tokens_b.extend(tokenization_function(token_b))
                example = example._replace(tokens_b=new_tokens_b if new_tokens_b
else [''])

        elif type(example).__name__ == 'SequenceLabellingExample':
            tokens = example.token_sequence
            labels = example.label_sequence
            assert tokens
            assert len(tokens) == len(labels)
            new_tokens, new_labels = [], []
            for token, label in zip(tokens, labels):
                retokenized_token = tokenization_function(token)
                if retokenized_token != [token]:
                    if label != 'O':
                        label_pos = label[:2]
                        label_type = label.split('-')[-1]
                        if label_pos == 'B-':
                            new_label = [label] + (len(retokenized_token) - 1) *
['I-' + label_type]
                        elif label_pos == 'I-':
                            new_label = [label] * len(retokenized_token)
                    else:
                        new_label = [label] * len(retokenized_token)
                    new_tokens.extend(retokenized_token)
                    new_labels.extend(new_label)
                else:
                    new_tokens.append(token)
                    new_labels.append(label)
            if new_tokens:
                example = example._replace(token_sequence=new_tokens)
                example = example._replace(label_sequence=new_labels)
```

```
            else:
                example = example._replace(token_sequence=[''])
                example = example._replace(label_sequence=['O'])
        examples[i] = example


def _truncate_seq_pair(tokens_a, tokens_b, max_length):
    """Truncates a sequence pair in place to the maximum length."""

    while True:
        total_length = len(tokens_a) + len(tokens_b)
        if total_length <= max_length:
            break
        if len(tokens_a) > len(tokens_b):
            tokens_a.pop()
        else:
            tokens_b.pop()

def convert_examples_to_features__tagging(
        args, tokenizer, examples, labels,
        pad_token_id, pad_token_label_id, max_seq_length):
    """Converts tagging examples into pytorch tensors."""

    InputFeatures = namedtuple(
        'SequenceLabelligFeatures',
        ['input_ids', 'input_mask', 'segment_ids', 'label_ids'])

    label_map = {label: i for i, label in enumerate(labels)}

    data_iterator = tqdm.tqdm(enumerate(examples), total=len(examples))

    features = []
    for i, example in data_iterator:
        tokens = example.token_sequence
        labels = example.label_sequence

        label_ids = []
        for token, label in zip(tokens, labels):
            if token.startswith('##'):
                label_ids.append(pad_token_label_id)
            else:
                label_ids.append(label_map[label])

        special_tokens_count = 2
        if len(tokens) > max_seq_length - special_tokens_count:
            tokens = tokens[: (max_seq_length - special_tokens_count)]
            label_ids = label_ids[: (max_seq_length - special_tokens_count)]

        tokens += ["[SEP]"]
        label_ids += [pad_token_label_id]
```

```
        segment_ids = [0] * len(tokens)

        tokens = ["[CLS]"] + tokens
        label_ids = [pad_token_label_id] + label_ids
        segment_ids = [0] + segment_ids

        if args.embedding == 'bert-base-uncased':
            input_ids = tokenizer.convert_tokens_to_ids(tokens)
        else:
            input_ids = tokenizer.as_padded_tensor([tokens],
maxlen=max_seq_length)[0]

        input_mask = [1] * len(tokens)

        padding_length = max_seq_length - len(input_mask)
        if args.embedding == 'bert-base-uncased':
            input_ids += [pad_token_id] * padding_length
        input_mask += [0] * padding_length
        segment_ids += [0] * padding_length
        label_ids += [pad_token_label_id] * padding_length

        assert len(input_ids) == max_seq_length
        assert len(input_mask) == max_seq_length
        assert len(segment_ids) == max_seq_length
        assert len(label_ids) == max_seq_length

        if i < 3:
            logging.info("*** Example ***")
            logging.info("tokens: %s", " ".join([str(x) for x in tokens]))
            logging.info("input_ids: %s", " ".join([str(x) for x in input_ids]))
            logging.info("input_mask: %s", " ".join([str(x) for x in input_mask]))
            logging.info("segment_ids: %s", " ".join([str(x) for x in
segment_ids]))
            logging.info("labels: %s", " ".join([str(x) for x in labels]))
            logging.info("label_ids: %s", " ".join([str(x) for x in label_ids]))

        features.append(
            InputFeatures(
                input_ids=input_ids,
                input_mask=input_mask,
                segment_ids=segment_ids,
                label_ids=label_ids)
        )
    return features


def build_features(
        args, split, tokenizer, examples, labels,
        pad_token_id, pad_token_label_id, max_seq_length):
```

```
    logging.info("Building features from data...")
    if args.task == 'sequence_labelling':
        func = convert_examples_to_features__tagging
    else:
        func = convert_examples_to_features__classification
    features = func(
        args, tokenizer, examples, labels,
        pad_token_id, pad_token_label_id, max_seq_length
    )

    # Convert to Tensors and build dataset
    if args.embedding == 'bert-base-uncased':
        all_input_ids = torch.tensor([f.input_ids for f in features],
dtype=torch.long)
    else:
        all_input_ids = torch.tensor([f.input_ids.tolist() for f in features],
dtype=torch.long)
    all_input_mask = torch.tensor([f.input_mask for f in features],
dtype=torch.long)
    all_segment_ids = torch.tensor([f.segment_ids for f in features],
dtype=torch.long)
    if args.task == 'sequence_labelling':
        all_label_ids = torch.tensor([f.label_ids for f in features],
dtype=torch.long)
    else:
        all_label_ids = torch.tensor([f.label_id for f in features],
dtype=torch.long)

    dataset = TensorDataset(all_input_ids, all_input_mask, all_segment_ids,
all_label_ids)
    return dataset
```

training.py

```
import os
import logging
import datetime
from numpy.core.fromnumeric import shape

import tqdm
import numpy as np
import sklearn.metrics as sklearn_metrics
import metrics.sequence_labelling as seqeval_metrics
import torch.functional as F

import torch
from torch.utils.data import SequentialSampler, RandomSampler, DataLoader
from transformers import AdamW, get_linear_schedule_with_warmup
```

```
from utils.misc import set_seed


def train(args, dataset, model, tokenizer, labels, pad_token_label_id):
    """ Trains the given model on the given dataset. """

    train_dataset = dataset['train']
    train_sampler = RandomSampler(train_dataset)
    train_dataloader = DataLoader(
        train_dataset,
        sampler=train_sampler,
        batch_size=args.train_batch_size)

    n_train_steps__single_epoch = len(train_dataloader) //
args.gradient_accumulation_steps
    n_train_steps = n_train_steps__single_epoch * args.num_train_epochs
    args.logging_steps = n_train_steps__single_epoch

    no_decay = ["bias", "LayerNorm.weight"]
    optimizer_grouped_parameters = [
        {
            "params": [p for n, p in model.named_parameters()
                       if not any(nd in n for nd in no_decay)],
            "weight_decay": args.weight_decay,
        },
        {
            "params": [p for n, p in model.named_parameters()
                       if any(nd in n for nd in no_decay)],
            "weight_decay": 0.0
        },
    ]
    optimizer = AdamW(optimizer_grouped_parameters, lr=args.learning_rate,
eps=args.adam_epsilon)
    scheduler = get_linear_schedule_with_warmup(
        optimizer,
        num_warmup_steps=int(args.warmup_ratio*n_train_steps),
        num_training_steps=n_train_steps
    )

    logging.info("***** Running training *****")
    logging.info("  Num examples = %d", len(train_dataset))
    logging.info("  Num Epochs = %d", args.num_train_epochs)
    logging.info(
        "  Total train batch size (w. accumulation) = %d",
        args.train_batch_size * args.gradient_accumulation_steps
    )
    logging.info("  Gradient Accumulation steps = %d",
args.gradient_accumulation_steps)
    logging.info("  Total optimization steps = %d", n_train_steps)
    logging.info("  Using linear warmup (ratio=%s)", args.warmup_ratio)
```

```
    logging.info("  Using weight decay (value=%s)", args.weight_decay)
    global_step = 0
    epochs_trained = 0
    steps_trained_in_current_epoch = 0

    tr_loss, logging_loss = 0.0, 0.0
    best_metric, best_epoch = -1.0, -1

    model.zero_grad()
    train_iterator = tqdm.trange(epochs_trained, int(args.num_train_epochs),
desc="Epoch")

    set_seed(seed_value=args.seed)
    for num_epoch in train_iterator:
        epoch_iterator = tqdm.tqdm(train_dataloader, desc="Iteration")
        for step, batch in enumerate(epoch_iterator):

            if steps_trained_in_current_epoch > 0:
                steps_trained_in_current_epoch -= 1
                continue

            model.train()
            batch = tuple(t.to(args.device) for t in batch)
            inputs = {
                "input_ids": batch[0],
                "attention_mask": batch[1],
                "token_type_ids": batch[2],
                "labels": batch[3]}

            outputs = model(**inputs)
            loss = outputs

            if args.gradient_accumulation_steps > 1:
                loss = loss / args.gradient_accumulation_steps

            loss.backward()

            tr_loss += loss.item()
            if (step + 1) % args.gradient_accumulation_steps == 0:
                torch.nn.utils.clip_grad_norm_(model.parameters(),
args.max_grad_norm)

                optimizer.step()
                scheduler.step()
                model.zero_grad()
                global_step += 1

                if global_step % args.logging_steps == 0:
                    results, _ = evaluate(
```

```
                        args=args,
                        eval_dataset=dataset["validation"],
                        model=model, labels=labels,
                        pad_token_label_id=pad_token_label_id
                    )

                    logging_loss = tr_loss
                    metric = results['f1']

                    if metric > best_metric:
                        best_metric = metric
                        best_epoch = num_epoch


                        if not os.path.exists(args.output_dir):
                            os.makedirs(args.output_dir)
                        model.save_pretrained(args.output_dir)
                        if 'character' not in args.embedding:
                            tokenizer.save_pretrained(args.output_dir)
                        torch.save(args, os.path.join(args.output_dir,
"training_args.bin"))
                        logging.info("Saving model checkpoint to %s",
args.output_dir)


    return global_step, tr_loss / global_step, best_metric, best_epoch


def evaluate(args, eval_dataset, model, labels, pad_token_label_id):
    """ Evaluates the given model on the given dataset. """

    eval_sampler = SequentialSampler(eval_dataset)
    eval_dataloader = DataLoader(
        eval_dataset,
        sampler=eval_sampler,
        batch_size=args.eval_batch_size)

    logging.info("***** Running evaluation *****")
    logging.info("  Num examples = %d", len(eval_dataset))
    logging.info("  Batch size = %d", args.eval_batch_size)
    eval_loss = 0.0
    nb_eval_steps = 0
    preds = None
    out_label_ids = None
    model.eval()
    for batch in tqdm.tqdm(eval_dataloader, desc="Evaluating"):
        batch = tuple(t.to(args.device) for t in batch)

        with torch.no_grad():
            inputs = {
```

```
                "input_ids": batch[0],
                "attention_mask": batch[1],
                "token_type_ids": batch[2],
                "labels": batch[3] }

            p_input = {
                "input_ids": batch[0],
                "attention_mask": batch[1],
                "token_type_ids": batch[2]}

            outputs = model.predict(**p_input)

            logits = outputs

        try:
            nb_eval_steps += 1
            if preds is None:
                preds = np.array(logits)
                out_label_ids = inputs["labels"].detach().cpu().numpy()
            else:
                preds = np.append(preds, logits, axis=0)
                out_label_ids = np.append(out_label_ids,
inputs["labels"].detach().cpu().numpy(), axis=0)
        except Exception as e:
            print('errr')


    label_map = {i: label for i, label in enumerate(labels)}
    if args.task == 'classification':
        preds_list = np.argmax(preds, axis=1)
        results = {
            "precision": sklearn_metrics.precision_score(out_label_ids, preds_list,
average='micro'),
            "recall": sklearn_metrics.recall_score(out_label_ids, preds_list,
average='micro'),
            "f1": sklearn_metrics.f1_score(out_label_ids, preds_list,
average='micro'),
            "accuracy": sklearn_metrics.accuracy_score(out_label_ids, preds_list),
        }
    else:
        out_label_list = [[] for _ in range(out_label_ids.shape[0])]
        preds_list = [[] for _ in range(out_label_ids.shape[0])]
        for i in range(out_label_ids.shape[0]):
            for j in range(out_label_ids.shape[1]):
                if out_label_ids[i, j] != pad_token_label_id:
                    out_label_list[i].append(label_map[out_label_ids[i][j]])
                    preds_list[i].append(label_map[preds[i][j]])

        results = {
```

```
            "precision": seqeval_metrics.precision_score(out_label_list,
preds_list),
            "recall": seqeval_metrics.recall_score(out_label_list, preds_list),
            "f1": seqeval_metrics.f1_score(out_label_list, preds_list),
        }

    logging.info("***** Eval results *****")
    for key in sorted(results.keys()):
        logging.info("  %s = %s", key, str(results[key]))

    return results, preds_list
```

conll.py

```
from ctypes import windll
import os

sentences = []
lines = []

class Dataset(object):
    def __init__(self, filename, processing_word=None, processing_tag=None,
                 max_iter=None):
        self.filename = filename
        self.processing_word = processing_word
        self.processing_tag = processing_tag
        self.max_iter = max_iter
        self.length = None


    def __iter__(self):
        niter = 0
        with open(self.filename) as f:
            words, tags = [], []
            for line in f:
                line = line.strip()
                if (len(line) == 0 or line.startswith("-DOCSTART-")):
                    if len(words) != 0:
                        niter += 1
                        if self.max_iter is not None and niter > self.max_iter:
                            break

                        yield words, tags

                        words, tags = [], []
                else:
                    ls = line.split(' ')
                    if len(ls) < 2:
```

```
                        print(ls)
                        continue
                    word, tag = ls[0],ls[-1]
                    if self.processing_word is not None:
                        word = self.processing_word(word)
                    if self.processing_tag is not None:
                        tag = self.processing_tag(tag)
                    words += [word]
                    tags += [tag]


    def __len__(self):
        """Iterates once over the corpus to set and store length"""
        if self.length is None:
            self.length = 0
            for _ in self:
                self.length += 1

        return self.length

with open('train.txt', 'a', encoding='utf8') as file:
    data = Dataset('eng.train')
    for s in data:
        for (word, tag) in zip(s[0], s[1]):
            file.write(f'{word} {tag}\n')

        file.write('\n')

with open('test.txt', 'a', encoding='utf8') as file:
    data = Dataset('eng.testa')
    for s in data:
        for (word, tag) in zip(s[0], s[1]):
            file.write(f'{word} {tag}\n')

        file.write('\n')

with open('validate.txt', 'a', encoding='utf8') as file:
    data = Dataset('eng.testb')
    for s in data:
        for (word, tag) in zip(s[0], s[1]):
            file.write(f'{word} {tag}\n')

        file.write('\n')
```

server.py

```
import flask
import predict
```

```
from flask import request, jsonify

app = flask.Flask("ner_python_api")
app.config["DEBUG"] = True

@app.route('/', methods=['POST'])
def home():
    body = request.json

    if 'text' in body:
        text = body['text']
    elif 'Text' in body:
        text = body['Text']
    else:
        return 'Error: No text provided!'

    return jsonify(predict.run(text))

app.run(port=11155)
```

Startup.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.OpenApi.Models;
using NerService.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace NerService
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }
```

```
        // This method gets called by the runtime. Use this method to add services
to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddScoped<NerPythonService>();
            services.AddHttpClient();

            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "NerService", Version
= "v1" });
            });
        }

        // This method gets called by the runtime. Use this method to configure the
HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseSwagger();
                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"NerService v1"));
            }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

NerPythonService.cs

```
using Microsoft.Extensions.Configuration;
using NerService.Data;
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;

namespace NerService.Services
{
    public class NerPythonService
    {
        private readonly HttpClient httpClient;
        private readonly IConfiguration configuration;

        public NerPythonService(HttpClient httpClient, IConfiguration
configuration)
        {
            this.httpClient = httpClient;
            this.configuration = configuration;
        }


        public async Task<IEnumerable<ResultDto>> GetAsync(string text)
        {
            var jsonConfig = new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            };

            var reqContent = new StringContent(
                JsonSerializer.Serialize(new RequestDto {Text = text}),
                Encoding.UTF8,
                "application/json");


            var url = configuration["NER:PythonEndpoint"];
            var response = await httpClient.PostAsync(url, reqContent);

            if (response is null || !response.IsSuccessStatusCode)
                throw new Exception("Server error!");

            var content = await response.Content.ReadAsStringAsync();
            var results = JsonSerializer
                .Deserialize<IEnumerable<NerResultDto>>(content, jsonConfig);

            return ParseResponse(results);
        }

        private IEnumerable<ResultDto> ParseResponse(IEnumerable<NerResultDto> ner)
        {
            var list = new List<ResultDto>();
```

```
            foreach (var result in ner)
            {
                var resultDto = new ResultDto { Senctence = result.Sentence };

                var labels = result.Labels;
                var tokens = result.Tokens;

                var label = string.Empty;
                var entTokens = new StringBuilder();

                var entites = new List<NerEntity>();

                foreach (var pair in tokens.Zip(labels))
                {
                    if (pair.Second.StartsWith("B"))
                    {
                        if (!string.IsNullOrEmpty(label))
                            entites.Add(new NerEntity
                            {
                                Label = label,
                                Tokens = entTokens.ToString()
                            });

                        label = pair.Second.Split('-').Last();
                        entTokens = new StringBuilder();
                        entTokens.Append(pair.First);
                    }
                    else if (pair.Second.StartsWith("I") &&
                             pair.Second.Split('-').Last() == label)
                        entTokens.Append($" {pair.First}");
                }

                if (!string.IsNullOrEmpty(label))
                    entites.Add(new NerEntity { Label = label, Tokens =
entTokens.ToString() });

                resultDto.Enitites = entites;

                list.Add(resultDto);
            }

            return list;
        }
    }
}
```

NerController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using NerService.Data;
using NerService.Services;
using System.Threading.Tasks;

namespace NerService.Controllers
{
    [ApiController]
    [Route("api/v1/[controller]")]
    public class NerController : ControllerBase
    {

        private readonly NerPythonService nerService;

        public NerController(NerPythonService nerService)
        {
            this.nerService = nerService;
        }

        [HttpPost]
        public async Task<IActionResult> Get([FromBody] RequestDto req)
        {
            return Ok(await nerService.GetAsync(req.Text));
        }
    }
}
```

ДОДАТОК Г.

Ілюстративна частина

РОЗРОБКА МЕТОДУ І ЗАСОБІВ ВЕБ-СИСТЕМИ ДЛЯ ПОШУКУ ІМЕНОВАНИХ СУТНОСТЕЙ У ТЕКСТІ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

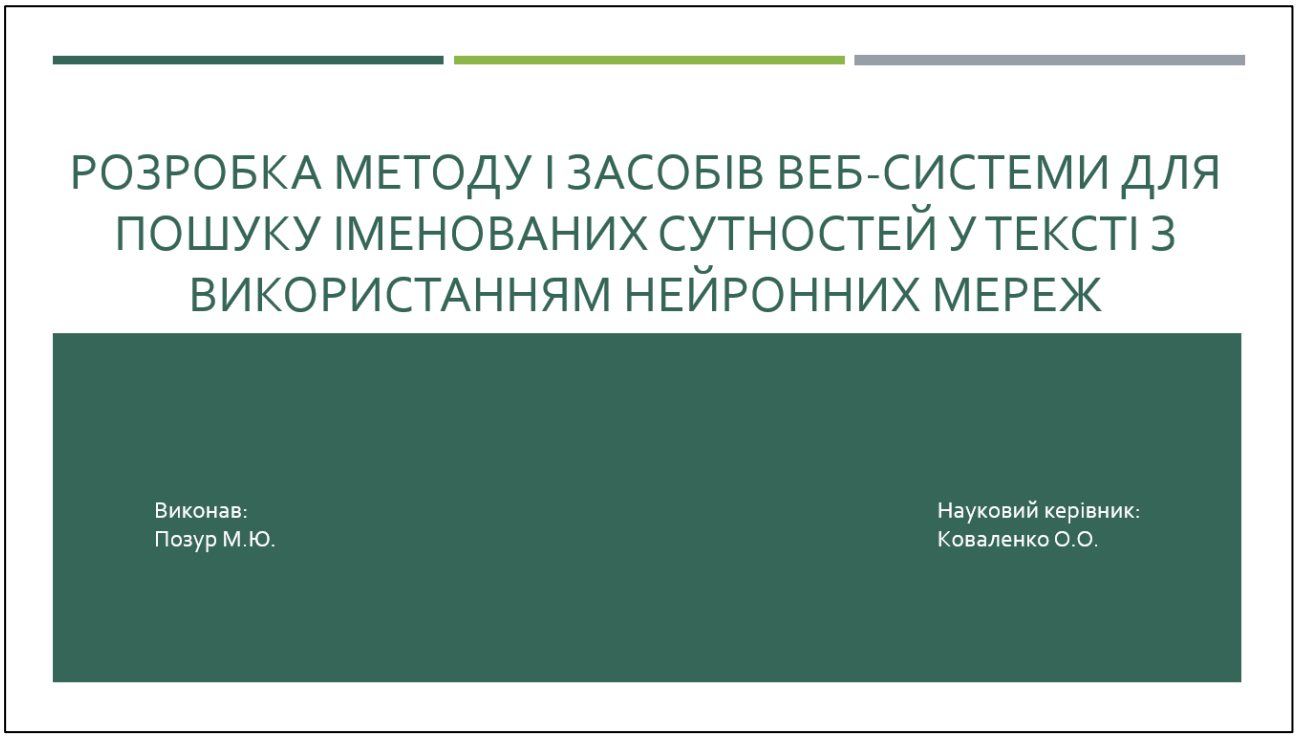

Рисунок Г.1 – Назва роботи

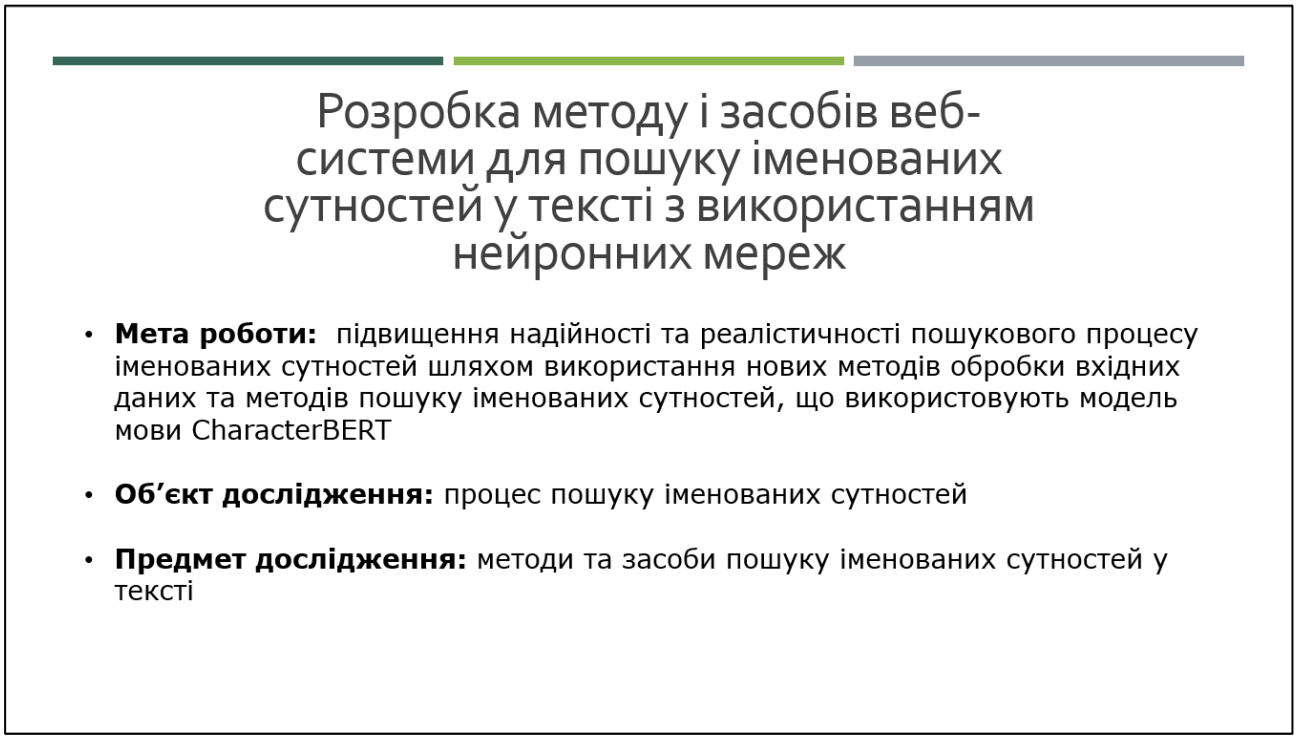

Рисунок Г.2 – Мета, об'єкт і предмет дослідження

Задачі

- визначити найбільш ефективний підхід до пошуку іменованих сутностей у тексті;
- розробити метод підготовки вхідних даних;
- розробити метод пошуку іменованих сутностей у тексті;
- розробити модуль пошуку іменованих сутностей;
- розробити веб-сервіс для роботи з модулем пошуку іменованих сутностей;
- провести тестування програмного додатку.

Рисунок Г.3 – Задачі

Аналоги

- Apache OpenNLP
- GATE
- Azure Cognitive Services
- Google Natural Language

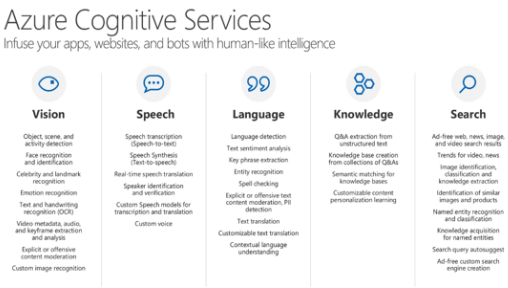

Рисунок Г.4 – Аналоги

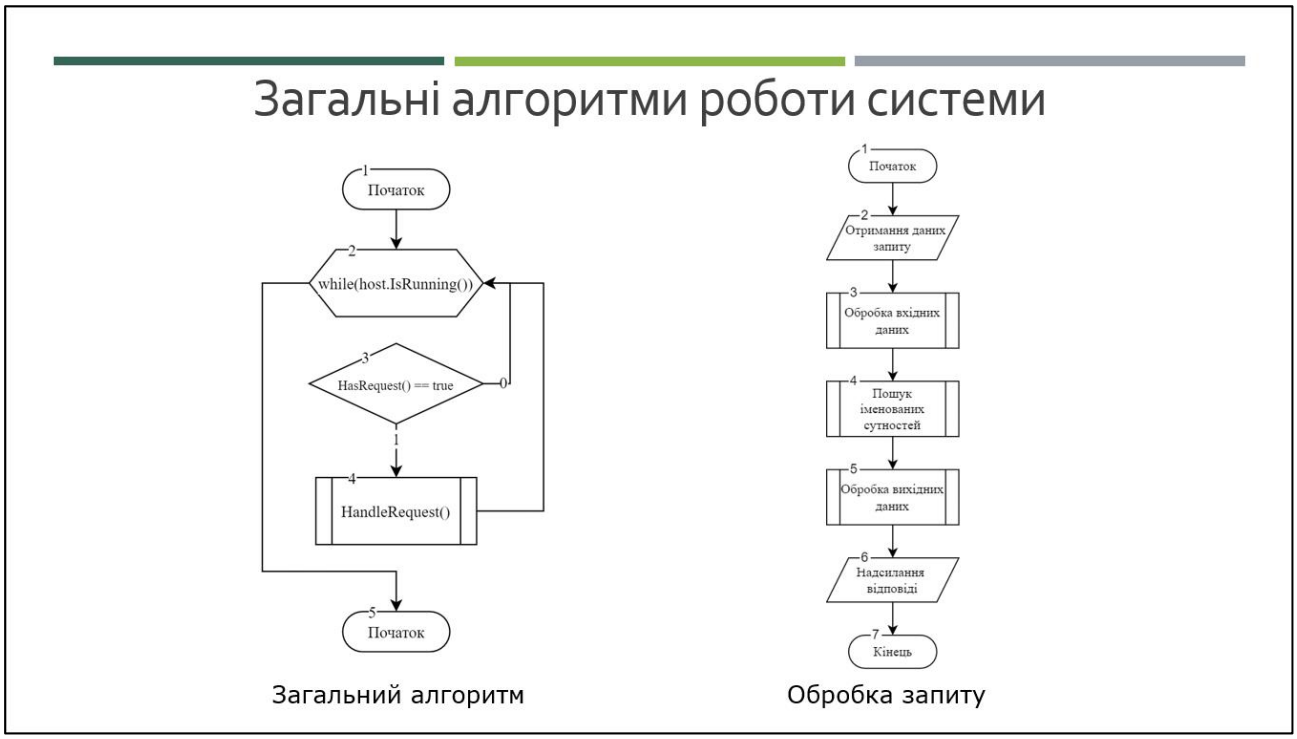

Рисунок Г.5 – Загальні алгоритми роботи системи

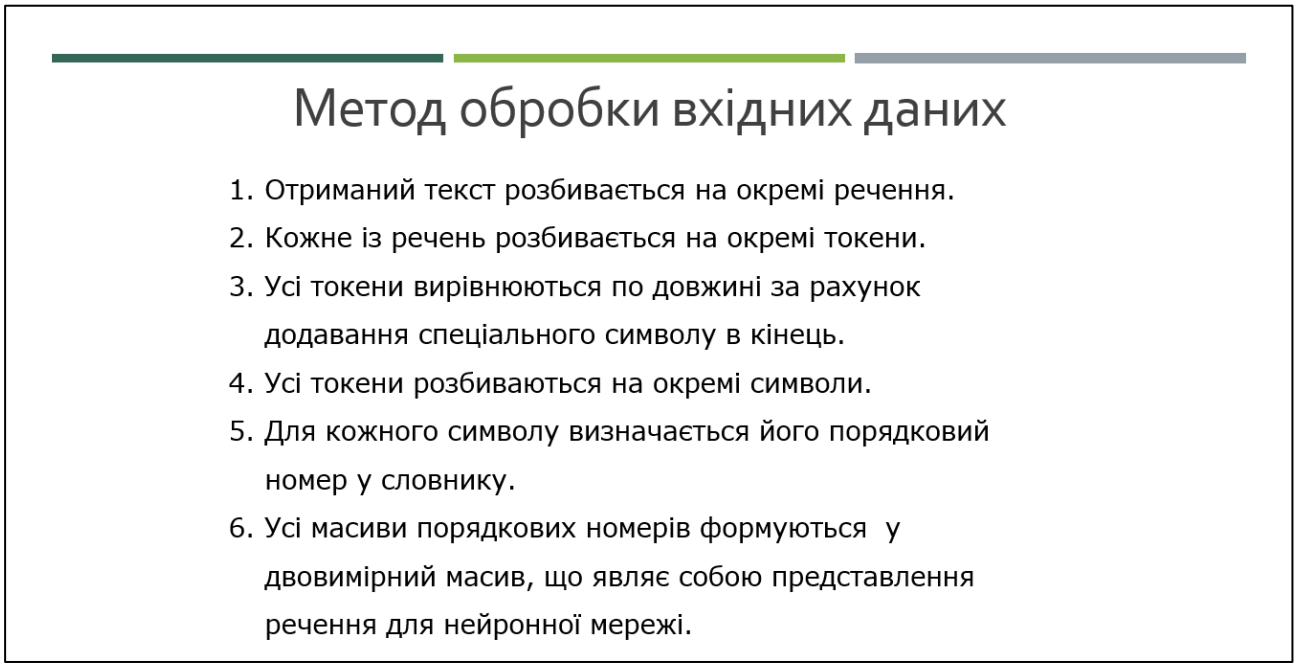

Рисунок Г.6 – Метод обробки вхідних даних

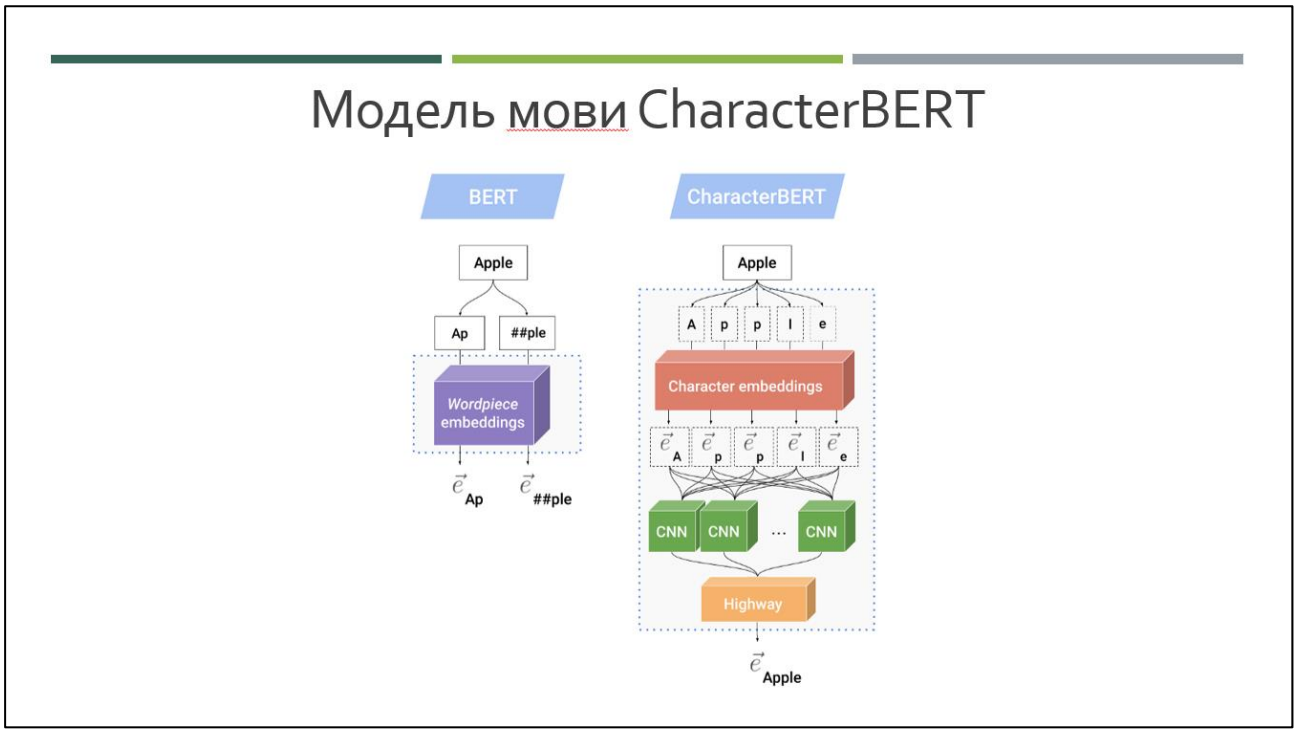

Рисунок Г.7 – Модель мови CharacterBERT

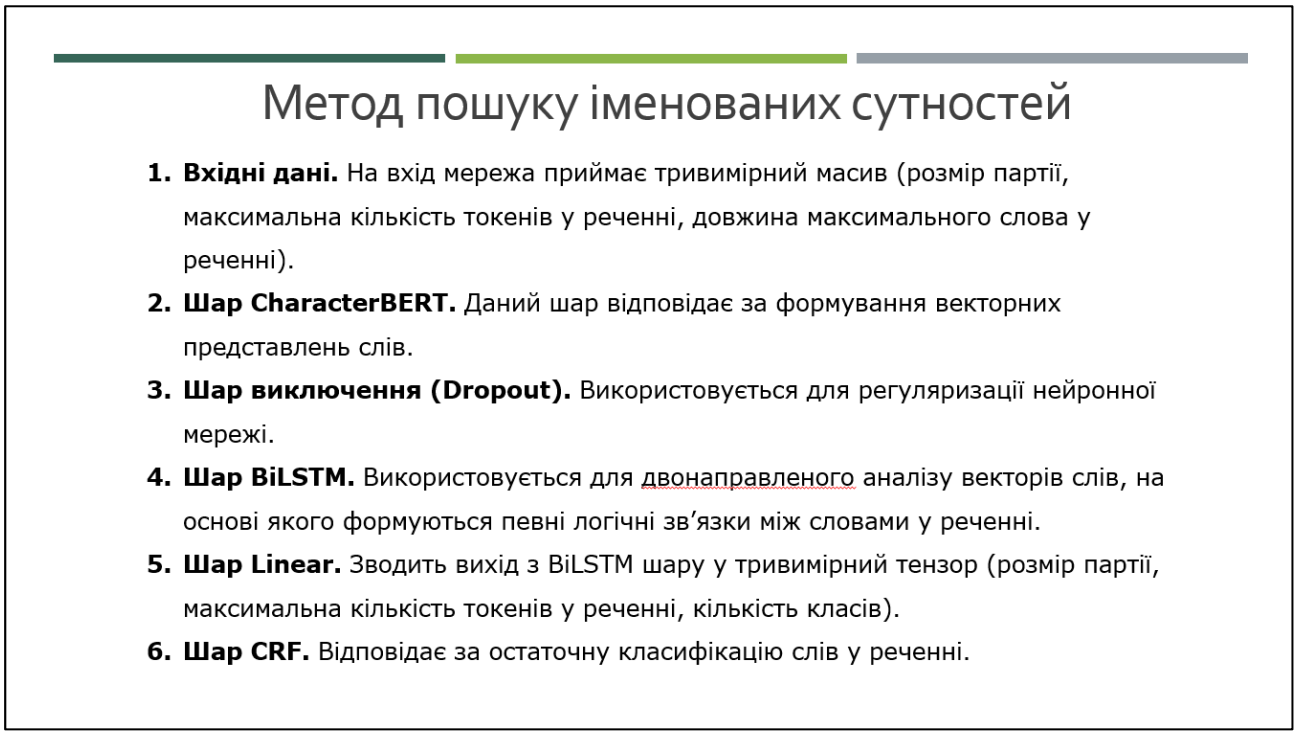

Рисунок Г.8 – Метод пошуку іменованих сутностей

Рисунок Г.9 – Структура нейронної мережі

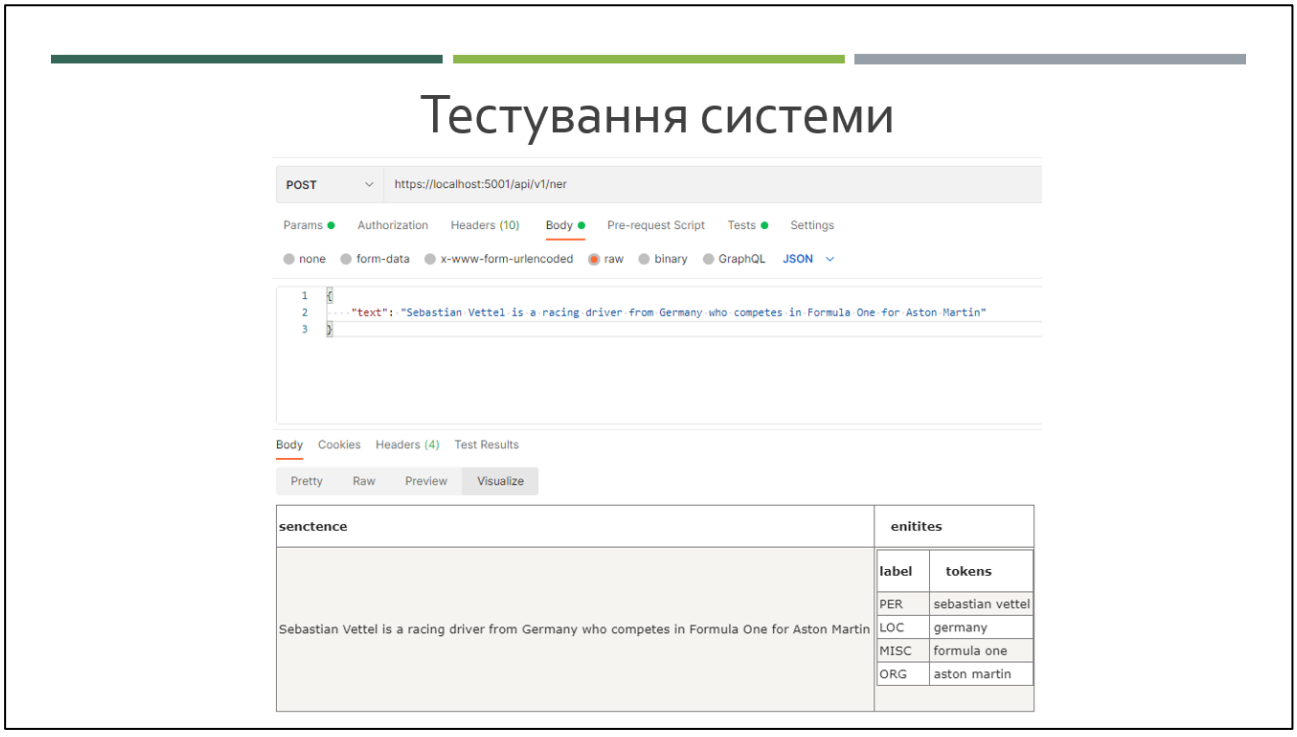

Рисунок Г.10 – Приклад тестування 1

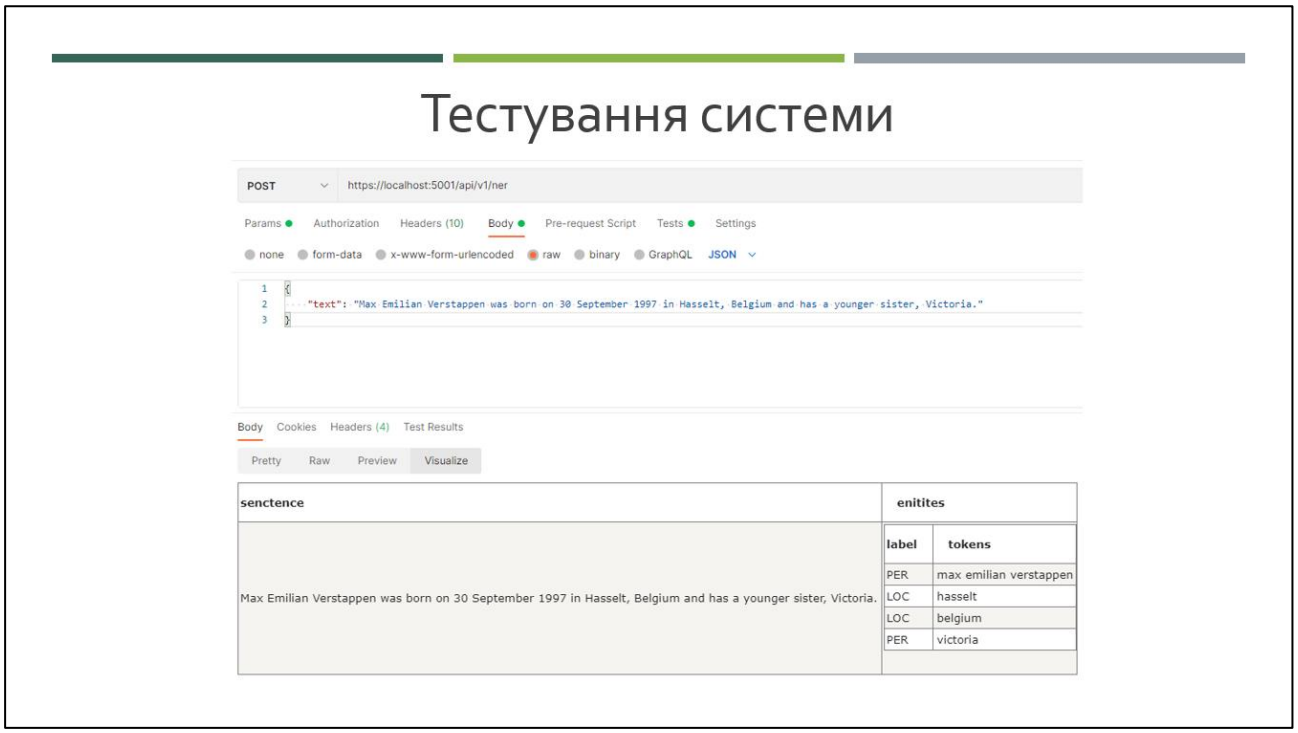

Рисунок Г.11 – Приклад тестування 2

- **Наукова новизна:**
  - подальшого розвитку отримав метод розпізнавання іменованих сутностей, що базується на використанні BERT-BiLSTM-CRF нейронної мережі, у якому, на відміну від існуючих, використовується модель мови CharacterBERT, чим вдається підвищити надійність роботи системи;
  - подальшого розвитку отримав метод підготовки даних до пошуку іменованих сутностей, у якому, на відміну від існуючих, текст розбивається на речення, що дозволяє дискретизувати вхідні дані й підвищити надійність пошуку іменованих сутностей.
- **Практична цінність:**
  - Практичне значення магістерської кваліфікаційної роботи полягає у можливості практичного використання розроблених алгоритмів і веб-сервісу для пошуку іменованих сутностей у процесі обробки текстової інформації

Рисунок Г.12 – Наукова новизна та практична цінність

Висновки

Під час виконання магістерської кваліфікаційної роботи було:

- визначено найбільш ефективний підхід до пошуку іменованих сутностей у тексті;
- розроблено метод пошуку іменованих сутностей у тексті та метод обробки вхідних даних;
- розроблено веб-сервіс для взаємодії із модулем розпізнавання іменованих сутностей;
- розроблено нейронну мережу для розпізнавання іменованих сутностей у тексті;
- проведено тестування системи.

Рисунок Г.13 – Висновки

Апробації

- **Результати магістерської кваліфікаційної роботи доповідалися та обговорювалися на:**
  - міжнародній науково-практичній конференції молодих вчених та студентів «Молодь у світі сучасних технологій» (Херсон 2020);
  - міжнародній науково-практичній інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова» (Вінниця 2021).

Рисунок Г.14 – Апробації

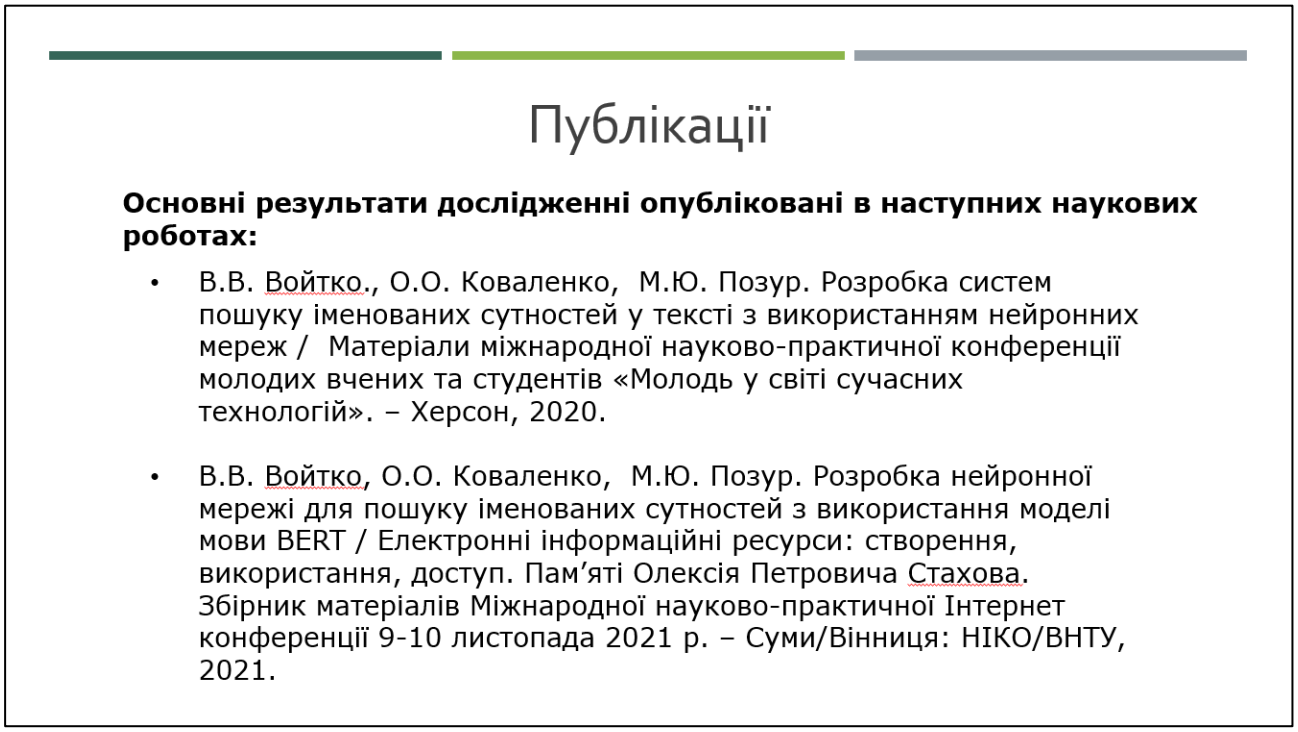

Рисунок Г.15 – Публікації