КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття рівня бакалавра
за спеціальністю
121 Інженерія програмного забезпечення
на тему:

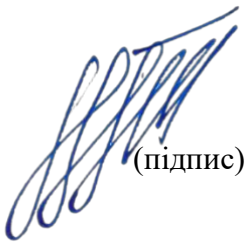
Розробка телеграм-бота парсингу сайтів з обробкою запитів природної мови

Виконав:
студент 4-го курсу
Павло Мікуш

(підпис)

Пропоную оцінку 98 (дев'яносто вісім)

Науковий керівник:
доцент, кандидат фізико-математичних наук
Володимир Шевченко



(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем

«25» травня 2022 р.,

протокол №10

Завідувач кафедри

Олександр ПРОВОТАР

_____
(підпис)

Київ – 2022

ЗМІСТ

РЕФЕРАТ

Кількість сторінок: 57

Кількість ілюстрацій: 19

Кількість таблиць: 2

Кількість джерел: 24

Кількість додатків: 2

КЛЮЧОВІ СЛОВА:

ТЕЛЕГРАМ-БОТ, ПАРСИНГ, ОБРОБКА ЗАПИТІВ ПРИРОДНОЇ МОВИ, PYTHON, NATURAL LANGUAGE PROCESSING, ХМАРНІ ТЕХНОЛОГІЇ, НЕЙРОННІ МЕРЕЖІ.

Об’єкт дослідження або розробки:

Розробка автоматизованих телеграм-ботів, які містять у своєму функціоналі парсинг інформації з певних веб-сайтів, а також класифікація даної інформації за певними категоріями за допомогою алгоритмів обробки запитів природної мови.

Мета кваліфікаційної роботи:

Розробити Telegram-бота для парсингу інформації з популярних соціальних мереж, зокрема Google, Pinterest, Reddit, Instagram та Twitter, з вбудованим в телеграм-бот алгоритмом класифікації даної інформації за певними категоріями.

Методи та інструменти дослідження або розробки:

- Мова програмування - Python з його додатковими бібліотеками
- Методи парсингу - парсинг інформації відбувається через аналіз HTML дерева веб-сторінки за допомогою програмних бібліотек *requests* та *beatifulsoup4,* а також запитів до API певних веб-сайтів.

- Методи обробки природної мови - нейромережеві алгоритми, зокрема такі як BERT. Реалізація цих алгоритмів відбувається за допомогою бібліотеки глибокого навчання *pytorch*
- Система керування базою даних - PostgreSQL
- Хмарне середовище – Genesis Cloud

Результати та їх новизна:

В даній бакалаврській роботі було розроблено телеграм-бот, що містять в собі парсинг та класифікацію даної інформації. Зауважимо, що даний бот функціонує на віддаленому сервері, що забезпечує його постійну неперервну роботу. Новизною роботи полягає у поєднанні є поєднання в одному телеграм-боті парсингу інформації з певних веб-сайтів та її класифікації за зручними для використання категоріями. Можливість практичного використання розробленого програмного забезпечення була підтверджена машинним експериментом.

Сфера застосування:

Застосування телеграм-ботів є надзвичайно широким, практично всі великі компанії на українському ринку мають власні телеграм-боти для своїх клієнтів. Як правило вони мають промисловий характер, проте в користувачів є широкий потреба у на зручних ботів для повсякденних використання, наприклад, для пошуку інформації. Саме в цьому аспекті і було проведено дослідження, проте окрім пошуку інформації, скрапінгу, було також реалізовано алгоритм класифікації для поділу веб-сторінок на певні категорії. Такий алгоритм також може бути корисний, зокрема, для новинних сайтів, яким буде зручно визначати категорії під якими випускати певну статтю або ж для класифікації постів у соціальних мережах.

ВСТУП

Сьогодні існує багато методів для різних етапів розробки телеграм-ботів, проте не до кінця досліджений аспект парсингу інформації та застосування алгоритмів обробки запитів природної мови в них.

Актуальність роботи та підстави для її виконання:

Одним з найпопулярніших та найстабільніших месенджерів на сьогодні є Telegram. Часто трапляються ситуації коли потрібно здійснити швидкий пошук інформації за певним запитом одразу у декількох соціальних мережах/веб-сайтах не виходячи з самого месенджеру, в кваліфікаційній роботі було вирішено саме таку проблему. Зокрема ввівши певний запит користувач може отримати релевантну відповідь, у вигляді посилань, на відповідні веб-сторінки в соціальних мережах, а також завдяки алгоритмам обробки запитів природної мови до кожного з таких посилань буде прикріплена певна категорія до якої він належить. Наприклад, за запитом “Верховна Рада України” в Google, ми очевидно отримаємо багато новин на політичні теми, тому більшість з таких статей будуть мати мітку “Політика”.

Мета й завдання роботи:

Розробити Telegram-бот для парсингу інформації з популярних соціальних мереж з використанням алгоритмів обробки текстів природної мови для подальшої класифікації.

Об’єкт, методи й засоби дослідження або розробки:

Чат-бот в месенджері Telegram для скрапінгу соціальних мереж з використанням алгоритмів обробки текстів природної мови для задачі класифікації. Телеграм-бот, модулі для скрапінгу та обробки контенту природньою природної мови були написані на мові програмування Python з використанням різних бібліотек. Зокрема також з використанням системи

керування базою даних PostgreSQL та завантаження на хмарний сервіс Genesis Cloud.

Можливі сфери застосування.

Розроблене програмне забезпечення має практичне застосування для швидкого отримання інформації за певним запитом одразу в різних соціальних мережах. Завдяки проведеній класифікації посилань здійснюється відсічення потенційно непотрібного контенту.

1. АНАЛІЗ ПРЕДМЕТУ ДОСЛІДЖЕННЯ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Опис середовища розробки - месенджера Telegram

Останнє десятиліття показало, що звичайні соціальні мережі стали пережитком минулого, на їх заміну прийшли – месенджери. Месенджери – це програмне забезпечення, яке надає можливості миттєвого листування, аудіо та відеозв'язку.

Одним з найбільш популярних месенджерів у світі є Telegram. За повідомленням його власника – Павла Дурова, на початок 2021 року його відвідує понад 500 мільйонів користувачів впродовж місяця [1], а також декілька останніх років він знаходиться на вершині рейтингу найзавантажуваніших додатків на смартфон.

Месенджер Telegram має ряд переваг відносно своїх конкурентів, зокрема деякі з них:

- Захищеність приватності. Одна з важливих переваг даного месенджеру – це надійне шифрування повідомлень. В Телеграм доступні секретні чати, які використовують шифрування типу "клієнт-клієнт", в той час як звичайні "хмарні чати" – "клієнт-сервер/сервер-клієнт", що не залишає сліду на серверах компанії та забезпечує повну анонімність [2].
- Висока швидкість доставки повідомлень. Оскільки сервери компанії розташовані в багатьох точках земної кулі – це забезпечує надійність роботи та швидку передачу інформації.
- Дозволяє зберігати необмежену кількість інформації. Телеграм не обмежує кількість текстових, фото або відеоповідомлень, які будуть зберігатись, проте в одному повідомленні можна прикріпити файл розміром до 2 гігабайт.

- Повна кросплатформеність. Доступний на найпопулярніших платформах, наприклад на Linux, Windows, Android, iOS, Windows Phone та інших у вигляді додатків, та у будь-яких операційних системах з веб-браузера.
- Дозволяє створювати публічні чати до 200 тисяч учасників.
- Користувацькі чат-боти. Месенджер дозволяє користувачам створювати чат-ботів на будь-який смак, для цього дає зручне Bot API.
- Групові відео та аудіодзвінки з можливістю трансляції екрану.
- Інтерфейс телеграму підтримується 19-ма найпопулярнішими в світі мовами.

1.2 Аналіз телеграм-ботів

Також саме завдяки телеграм-ботам багато людей вподобали Telegram, а свідченням довіри до нього є офіційні телеграм-боти державних структур, зокрема Укрзалізниці [3], котра використовує його для онлайн-продажу квитків, а також приватних структур для прикладу віртуального банку - monobank [4].

Загалом телеграм-боти мають широкі можливості [5]:

- Надсилати повідомлення в будь-які чати, де вони додані. Це не лише особистий чат з ботом, а і в канали та публічні чати.
- Мають доступ до усіх типів повідомлень, включно з голосуваннями, файлами, аудіо та відеоповідомленнями та керуються, зокрема за допомогою запрограмованих його власником команд.
- Використовувати засоби для банківської оплати.
- Запрошувати у користувача інформацію про контактні дані (номер телефону), геолокацію та інше.

- Забезпечується спеціальний режим для “ігрових ботів”.

1.3 Парсинг як метод обробки інформації з веб-сайтів

Парсинг інформації - це процес отримання інформації в одному форматі та переведення її у інший потрібний формат. Наприклад, парсинг інформації відбувається в усіх компіляторах, коли нам потрібно перевести комп’ютерний код в машинний для подальшого виконання програми.

В еру експоненціального збільшення кількості даних під парсингом інформаціі зазвичай розуміється веб-парсинг. Тобто витягнення корисної інформації з веб-сторінок.

Глобально більшість веб-парсерів працюють за такою схемою:

1. Спочатку потрібно зберегти інформацію HTML-структури нашої сторінки. В ній зберігається вся інформація, яка доступна на сторінці.
2. Згодом ми уже працюємо з HTML-структурою. Переводимо її у зручний формат “дерева”, а звідти дістаємо потрібну нам інформацією.

Значна частина веб-сайтів перешкоджає автоматизованому збору інформації. Наприклад, блокують IP адреси, які вже були помічені в парсингу інформації, виявляють ботів через їхню “user agent” стрічку, яка описує бота який зайшов на сайт, реагує на високу швидкість взаємодії з веб-сайтом, що характерно для програмного забезпечення написаного для парсингу інформації тощо.

Зазначимо, що деякі сайти забороняють парсинг своєї інформації. Також законодавством карається збір інформації, який може завдати збитків веб-ресурсам.

1.4 Огляд завдань обробки природної мови

NLP (Natural Language Processing - обробка природної мови) - це одна з підгалузей штучного інтелекту, яка відповідає за роботу з текстовими даними.

Деякі основні завдання NLP наведені в таблиці 1.

Назва завдання	Приклади застосувань
Машинний переклад	Grammarly, Google перекладач
Перевірка коректності текстів	Grammarly; вбудовані методи коректування в редактори тексту, зокрема Microsoft Word, Google Docs та інші
Класифікація текстів	Класифікація текстів на спам в поштових скриньках, а також класифікація небезпечних постів в соціальних мережах
Суммаризація (пошук головних аспектів в певному тексті) тексту	Більшість пошуковиків, таких як Google, Yahoo, Bing використовують для видачі стисненого результату вмісту статей
Генерація тексту	Для покращення роботи чат-ботів на основі штучного інтелекту
Аналіз тексту для подальшого показу реклами	Використовується популярними сервісами типу Google Ads

Аналіз настроїв в тексті	Використовується соціальними мережами задля запобігання поширення насильницького контенту

Таблиця 1 Опис завдань обробки природної мови (NLP)

Завдання бакалаврської роботи - це завдання багатоміткової класифікації. Тобто коли вхідний текст класифікується одразу за декількома категоріями.

1.5 Огляд існуючих телеграм-ботів для парсингу інформації

В процесі пошуку, повних аналогів нашої бакалаврської роботи не було знайдено. Окремо існують рішення для парсингу інформації з різних сайтів, проте вони не надають міток по окремо наданому посиланню. Також існують боти, які дають передбачення по написаному тексті, проте зазвичай вони вирішують задачу класифікації «настрою» тексту, використовують прості моделі, що не гарантують хорошу точність та є одномовними.

1.5.1 LyBot - бот для пошуку музики в YouTube Music

LyBot [6] - телеграм-бот для зручного пошуку пісень в сервісі YouTube Music. Інтерфейс бота доступний на 11 мовах, одна з яких українська.

Функціонал даного бота дозволяє ввести будь-який запит для пошуку, навіть по тексту пісні, яку бажає знайти користувач. Після цього бот спарсить інформацію з YouTube Music та виведе назви пісень в зручному форматі. Згодом після натискання, користувачу буде надіслана відповідна композиція, яку він зможе завантажити на власний пристрій.

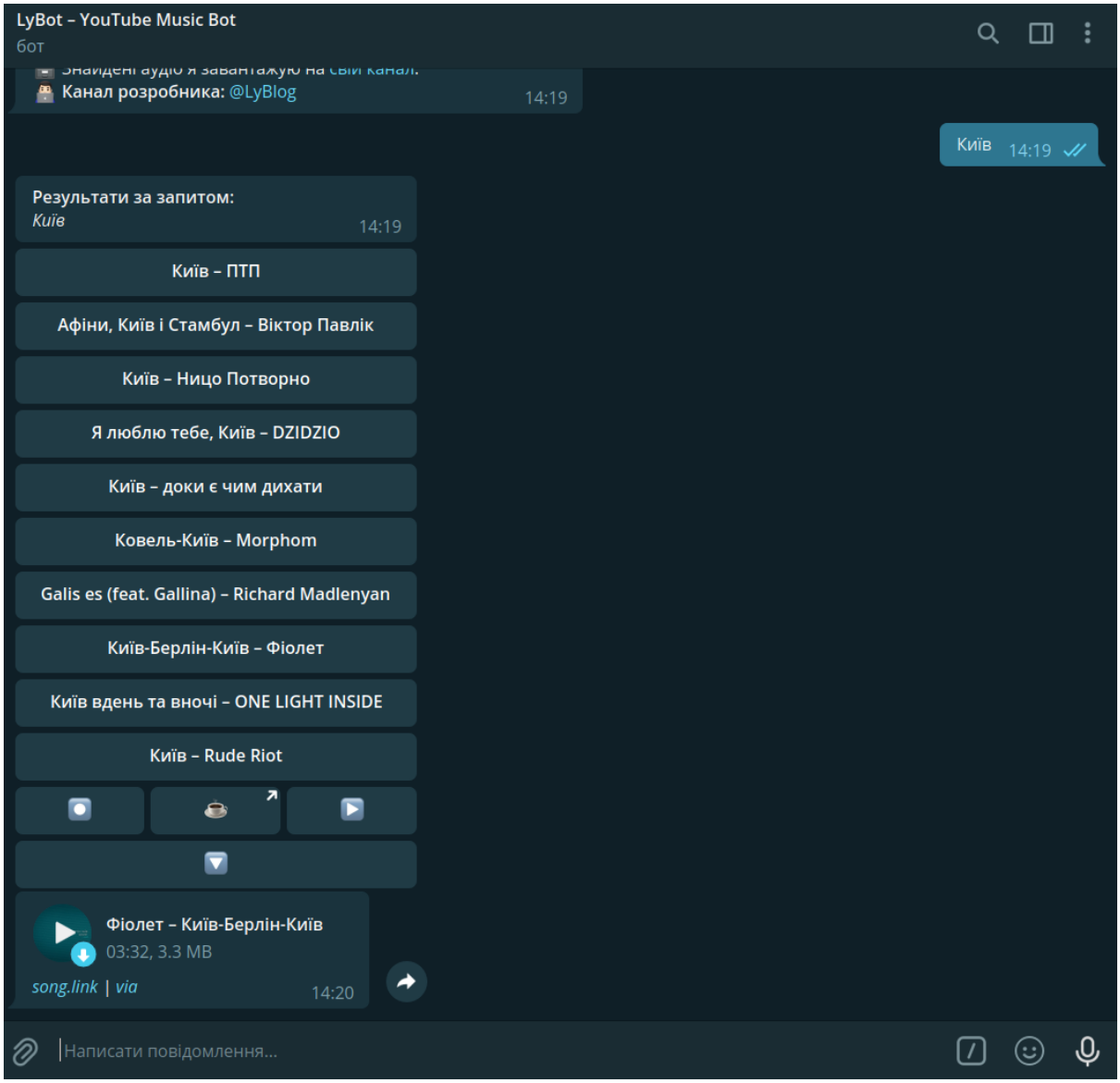


Рисунок 1 Приклад використання функціоналу LyBot

1.5.2 GoogleBot - парсинг посилань з Google

GoogleBot [7] - чат-бот, який дозволяє в зручній формі займатись парсингом посилань з Google пошуковика.

Через зручне інлайн-меню ми можемо знайти потрібне нам посилання та отримати його. Також бот доступний для надсилання посилань у групові чати та приватні чати.

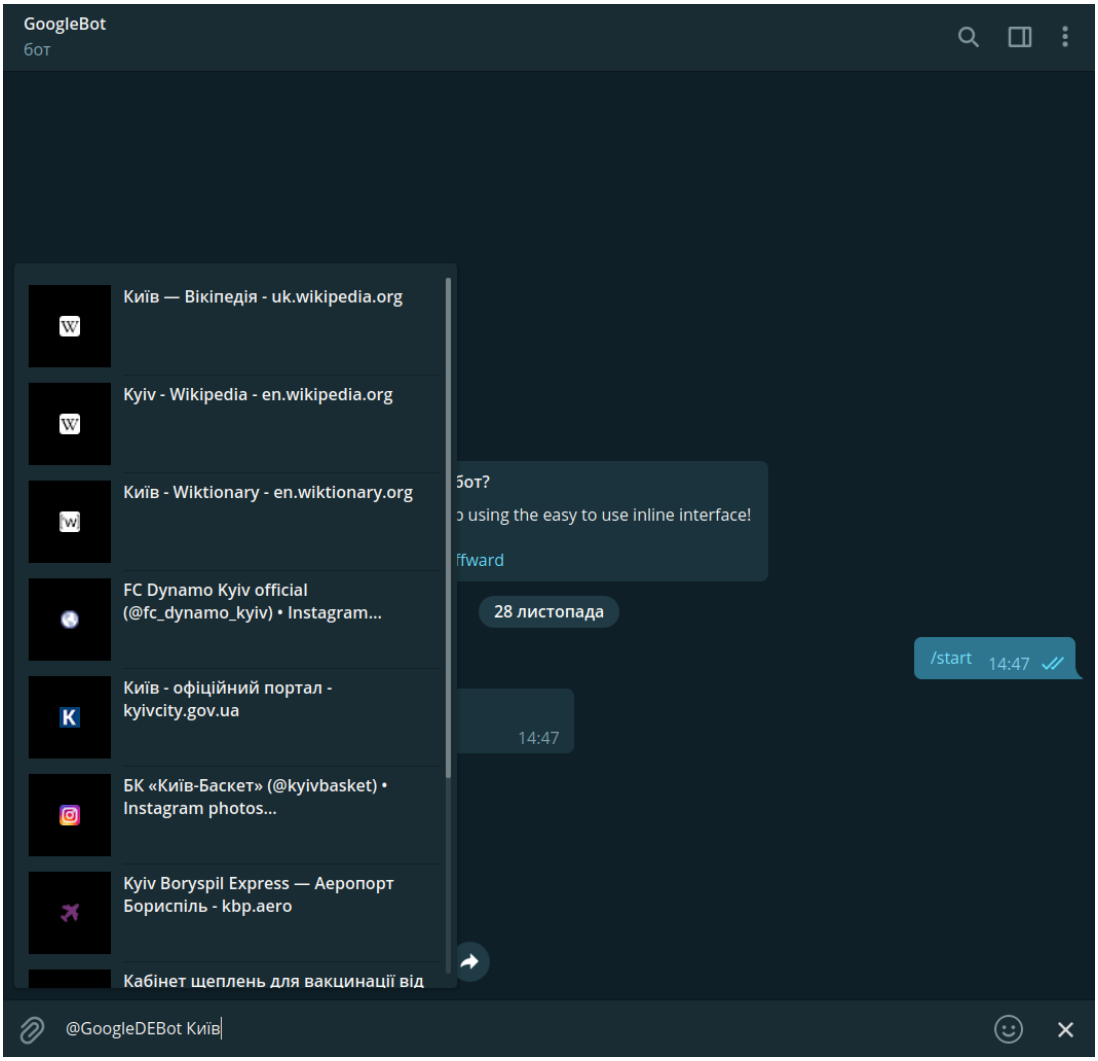


Рисунок 2 Приклад використання функціоналу GoogleBot

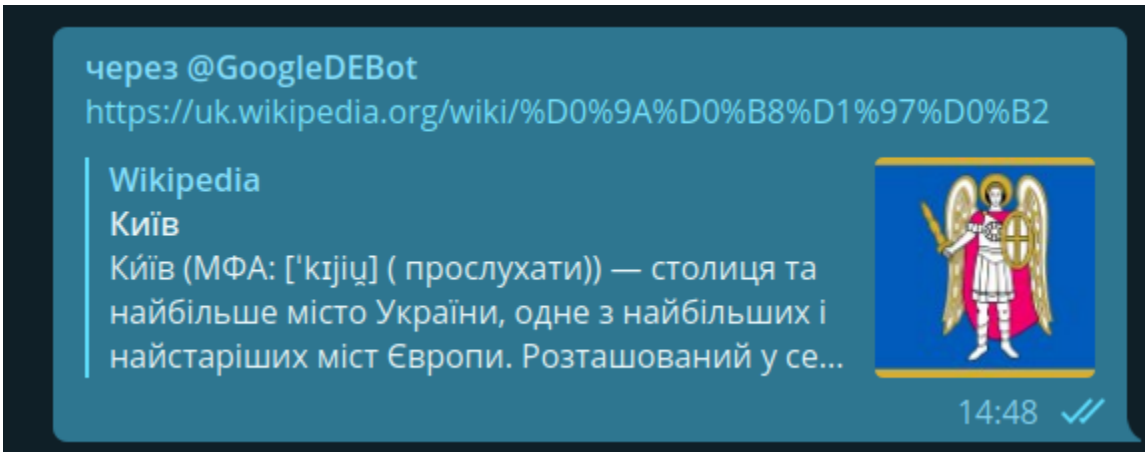


Рисунок 3 Приклад надсилання посилань по запиту

1.5.3 All Saver Bot - парсинг інформації з Youtube та Instagram.

All Saver Bot [8] - чат-бот, який дозволяє за посиланням завантажити відео з Instagram та Youtube. Доступний на трьох мовах, зокрема англійською.

Після введення потрібного посилання нам пропонують завантажити відео в форматі *mp4,* або ж у форматі *mp3*.

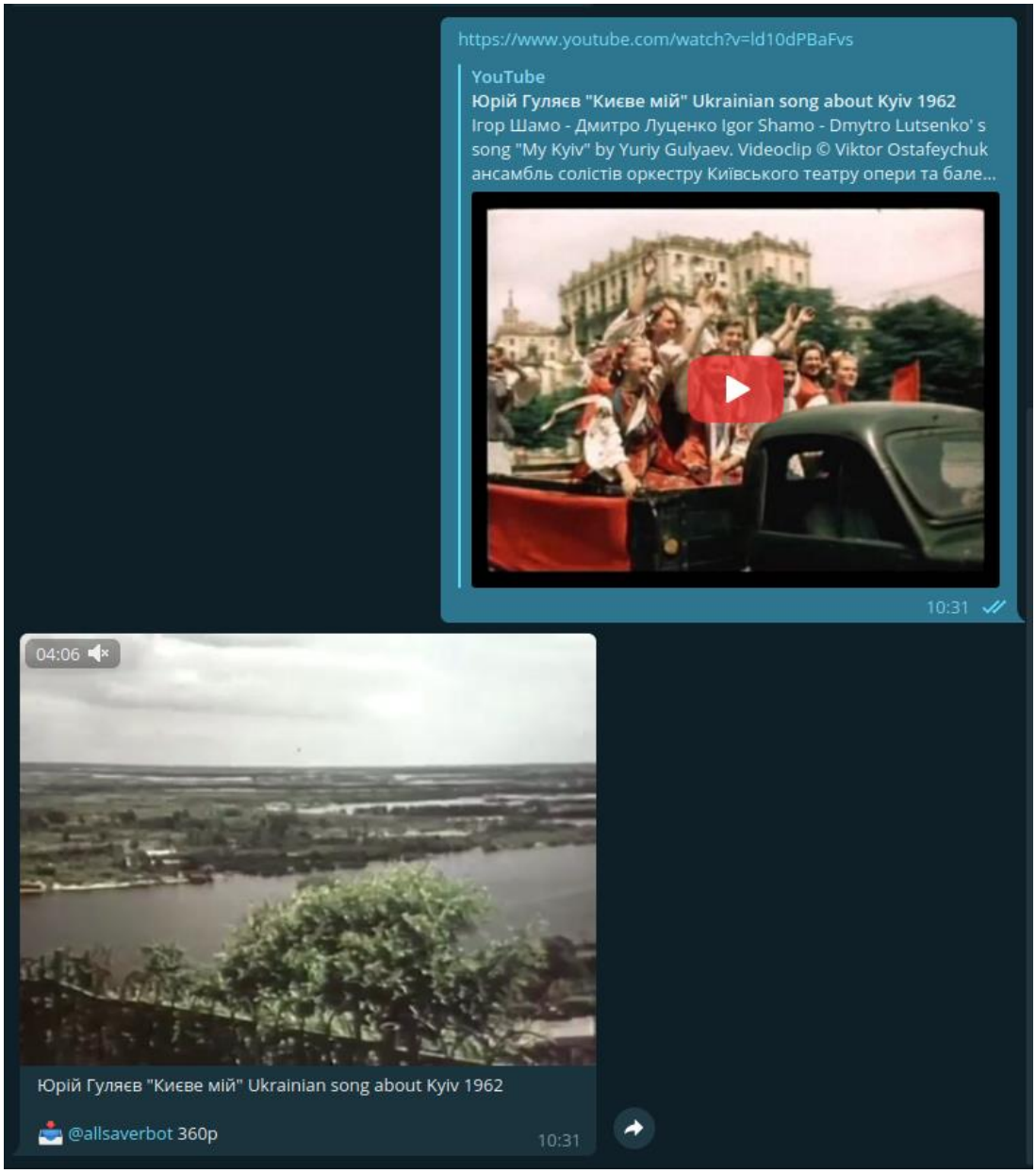


Рисунок 4 Приклад функціонування бота для Instagram

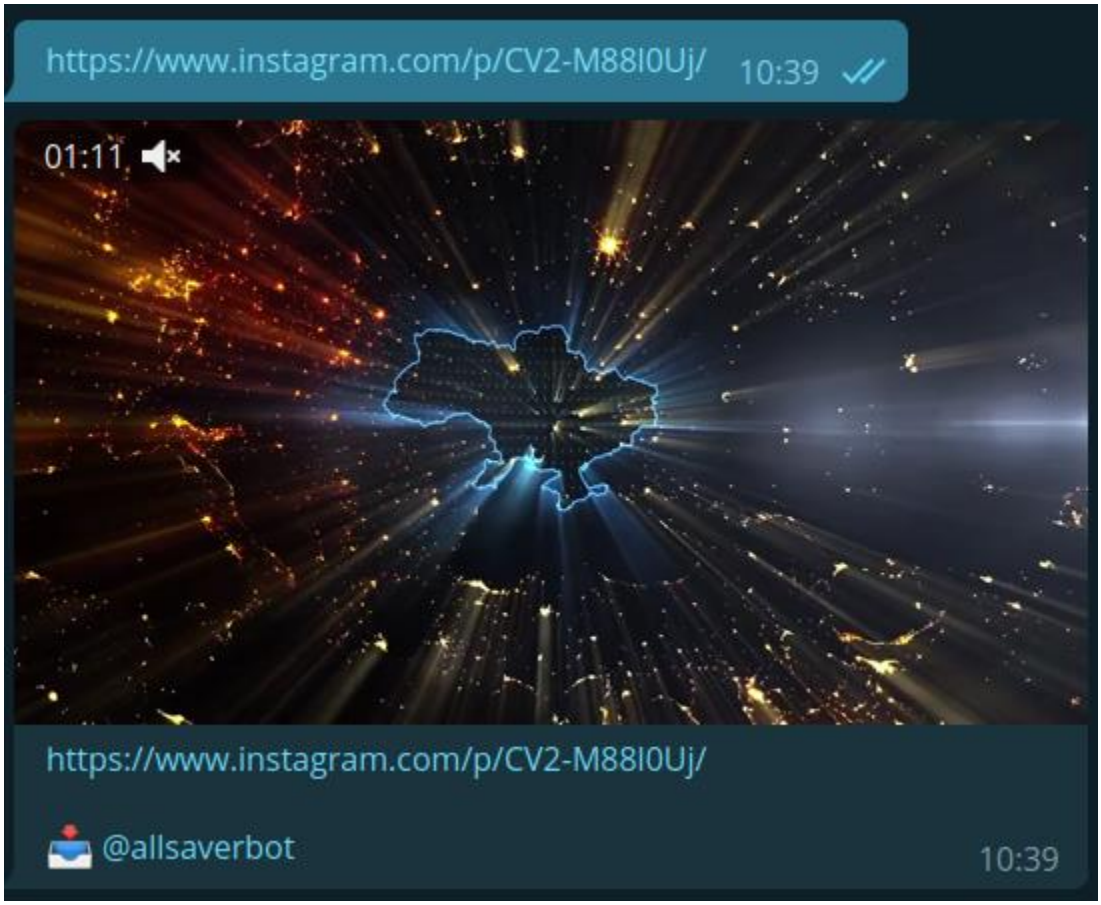


Рисунок 5 Приклад функціонування бота для Youtube

2. ФУНКЦІОНАЛЬНИЙ ОПИС

2.1 Опис програмного застосунку

Програмний застосунок можна описати через схему зображену на рисунку 6.

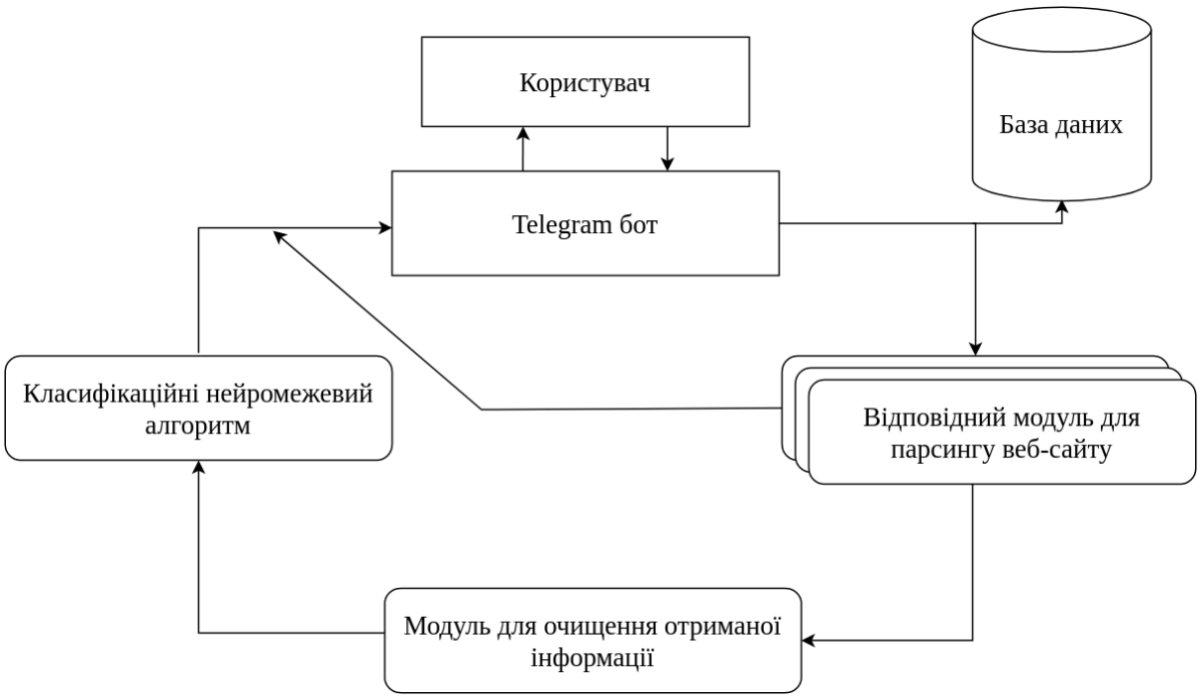


Рисунок 6 Структура програмного застосунку

Опишемо компоненти більш детально:

- Телеграм-бот. Використовується в якості інтерфейсу між користувачем та програмним функціоналом. Телеграм-бот забезпечений засобами зручного фронт-енду, зокрема у вигляді *inline* меню-кнопок.
- База даних. В базу-даних записується інформація про поточний вибір користувача, а саме пошуковий запит, вибрана соціальна мережа, мова користування (українська або англійська) та кількість постів котра була виведена.
- Відповідний модуль для парсингу соціальної мережі. Парсинг кожної веб-мережі унікальний, оскільки залежить від її веб-верстки. Програмний застосунок на даному етапі обирає за

вибором користувача парсинг якої соціальної мережі здійснюється. В розробленому програмному застосунку реалізовано методи парсингу для раду соціальних мереж, зокрема Pinterest, Twitter, Google, Reddit та Instagram.

- Модуль для очищення отриманої інформації. Після того, як ми отримали інформацію з попереднього етапу, її потрібно очистити перш ніж передавати в нейромежевий алгоритм. Оскільки інформація, котра приходить після парсингу часто має засмічений вигляд. Вона містить так звані стоп-слова, слова які не несуть ніякого лексичного забарвлення, може містити залишки HTML-дерева після парсингу. Нейромережевий алгоритм показує значно кращі результати на «дестильованих» даних, аніж на реальних.
- Класифікаційний нейромережевий алгоритм. Даний алгоритм відповідає за класифікацію наданого очищеного тексту за такими категоріями:
    - Бізнес
    - Політика
    - Розваги
    - Техніка
    - Спорт

Після того як алгоритм дав одноміткове передбачення, програмний застосунок повертає в бот посилання, які ми отримали на етапі парсингу з наявними мітками.

2.2 Опис бази даних

Для програмного застосунку описаного в підрозділі 2.1 було спроектовано реляційну базу даних, яка складається з двох таблиць.

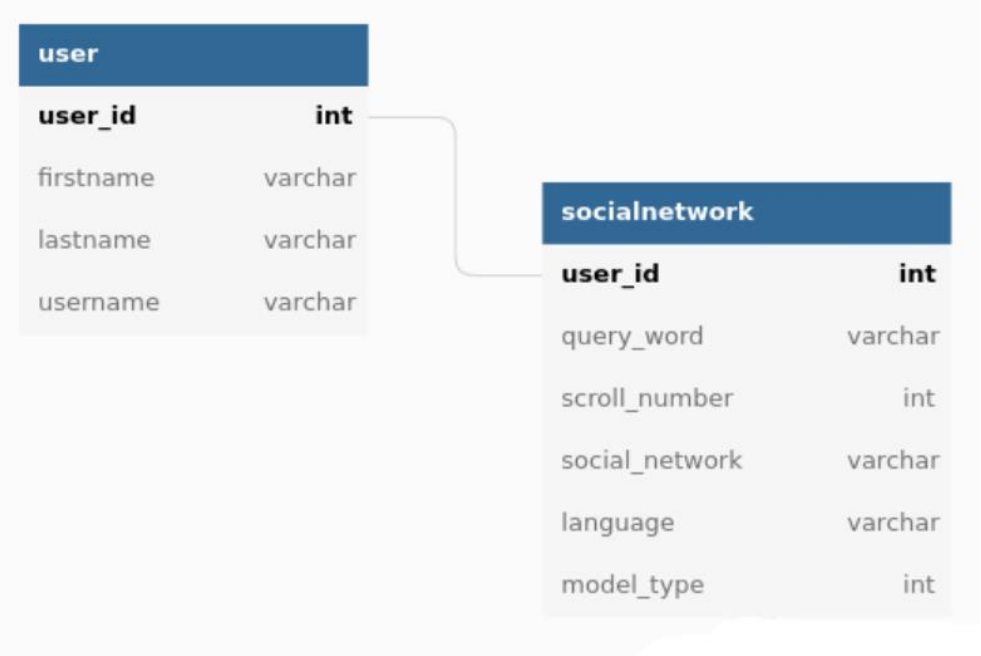


Рисунок 7 Схема бази даних створена за допомогою сервісу *dbdiagram.io*

telegramuser – таблиця котра відповідає за персональну інформацію про конкретного користувача

firstname – ім’я користувача

lastname – прізвище користувача

username – телеграм тег користувача

socialnetwork – таблиця котра відповідає за вибрану соціальну мережу:

user_id – унікальний ідентифікатор користувача Telegram

social_network – обрана користувачем соціальна мережа

query_word – пошуковий запит користувача

scroll_number – кількість посилань, які користувач виводив раніше

language – мова користування ботом

model_type – тип класифікаційного алгоритму. 1 означає використання алгоритму BERT, 0 – алгоритму на основі шарів LSTM.

Варто зауважити, що поле *user_id* є зовнішнім ключем на поле *user_id* в таблиці *telegramuser* з параметром DELETE CASCADE, що означає автоматичне видалення інформації в таблиці *socialnetwork* при видаленні відповідного ключа в таблиці *telegramuser*.

Оскільки нам не потрібно зберігати інформацію про виведені посилання, їх категорії, опис тощо, то заявленої структури бази даних більш ніж достатньо.

3 ІНСТРУМЕНТАЛЬНІ ЗАСОБИ

3.1 Взаємодія з Bot API

Зручне керування ботами Телеграм забезпечує через доступ до Телеграм API - Bot API, який представляє собою HTTP-інтерфейс.

Для початку взаємодії з телеграм-ботом його потрібно створити. Процес створення чат-бота в телеграмі є інтуїтивно зрозумілим та складається з декількох етапів:

1. Створення телеграм-бота розпочинається з його реєстрації в спеціальному мета-боті - BotFather (*@BotFather*). Для того щоб детальніше отримати інформацію про команди доступні в даному боті потрібно написати команду */help*.

2. Згодом за допомогою простої команди */newbot* ми реєструємо нового бота, даємо йому нове ім’я.

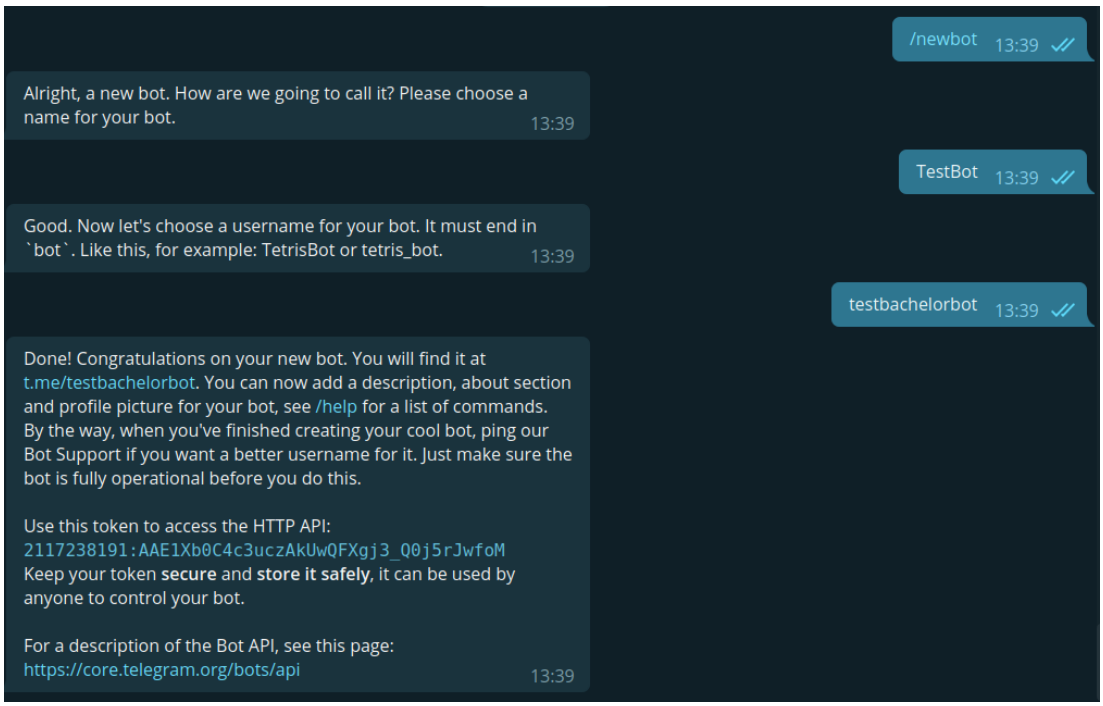


Рисунок 8 Реєстрація телеграм-бота

Після створення бота ми отримаємо його токен - це ключ авторизації для нашого телеграм-бота. Саме завдяки даному токену ми будемо надсилати запити до Bot API.

Важливо зберігати токен в таємниці, оскільки завдяки ньому зловмисники можуть перехопити контроль над ботом. Якщо ж користувач бажає змінити токен, то зробити це можна за допомогою команди */revoke*. Після цього користувачу буде видано новий токен, а старий буде деактивовано.

Запити до Bot API відбуваються через HTTPS в формі:

https://api.telegram.org/bot<token>/METHOD_NAME

Проте замість того, щоб взаємодіяти напряму з Bot API - це можна зробити за допомогою бібліотек в різних мовах програмування, які можуть надати для цього зручний інтерфейс.

3.2 Мова програмування Python

Розробку телеграм-ботів прийнято вести на скриптових мовах програмування, найчастіше зустрічаються: Python, Ruby, Node.js та PHP.

В ході дослідження бакалаврської роботи було обрано мову програмування Python, версії 3.8.

Python - високорівнева інтепритована об’єктно-орієнтована мова програмування з динамічною типізацією. Написаний Python на мові програмування C у 1989 році, за цей час став однією із найпопулярніших мов програмування у світі.

Однією з головних переваг Python є його розширюваність та велика кількість бібліотек під різні завдання. Окрім написаних бібліотек на самому Python, також для пришвидшення швидкодії

використовують вставки на мові програмування С. Переглянути такі бібліотеки можна на сайті *https://pypi.org/* та завантажити за допомогою команди *pip3 install name_of_package* у терміналі.

3.3. Бібліотеки мови програмування Python

3.3.1 Telebot

Telebot [9] - бібліотека є оболонкою над HTTP-запитами до Telegram Bot API, використовується для спрощення роботи та мінімізації кількості коду.

Для початку роботи з телеграм-ботом, спочатку потрібно оголосити екземпляр класу *telebot.TeleBot* з відповідним токеном для бота.

Бібліотека також забезпечує так звані “хендлери”, декоратори котрі створені для відслідковування типу надісланих повідомлень. Як і звичайні декоратори вони зазначаються над функцією, котра буде працювати з повідомленням даного формату:

@bot.message_handler(SPECIAL_FORMAT)

Типи форматів хендлерів [9] розміщені в таблиці 2.

Назва хендлера	Використання
content_types	Для різних форматів повідомлень, котрі надіслані в бота
regexp	Фільтрація повідомлень за допомогою регулярних виразів
commands	Команди, які наявні в боті. Зазвичай, це */start, /help, /info*
chat_types	Тип чату в якому бажаємо фільтрувати

	повідомлення
func	Фільтрація за допомогою довільної користувацької функції

Таблиця 2 Формати хенделерів для фільтрації повідомлень

Для забезпечення постійного функціонування телеграм-бота потрібно прописати, так званий Polling. Прописується вкінці коду бота та виглядає наступним чином:

bot.polling(none_stop=True)

Чат-боти повинні отримувати повідомлення від соціальної мережі моментально. Проте, перевіряти постійно оновлення неефективно, оскільки більшість відповідей від сервера будуть про те що оновлень немає. Саме цю проблему вирішує Polling, він *n* раз за секунду опитує сервер про зміни, які відбулись. Параметр *none_stope=True*, вказує на те що при виникненні будь-яких помилок зі сторони сервера або клієнта бот не повинен припиняти свою роботу.

Для надсилання повідомлень від бота ми маємо метод *bot.send_message*, також у бота є можливість редагувати його попередні повідомлення за допомогою методу *bot.edit_message*. Інші можливості даної бібліотеки детальніше описані в її документації [9].

3.3.2 Бібліотеки для парсингу інформації

В даному підрозділі описані основні бібліотеки мови програмування Python, які були використані в бакалаврській роботі.

3.3.2.1 Requests

Requests [10] - бібліотека, яка дозволяє надсилати HTTP запити через Python. Після того, як ми надіслали HTTP запит на певний сайт, ми можемо отримати в форматі структури даних - JSON, всю інформацію по даному посиланню.

За допомогою *get* методу можемо зробити простий запит на певний веб-сайт:

response = requests.get(URL_SITE)

Тоді в *response* зберігається екземпляр класу *Response*, який містить багато корисних атрибутів, зокрема деякі з них: *Response.status_code* - повертає статус нашого запиту (200-299 - для прикладу, якщо запит відбувався успішно та 500-599 - якщо відбулась помилка зі сторони серверу), *Response.content* в байт-коді повертає вміст сторінки, *Response.url* - посилання по якому здійснювався запит. Також за допомогою параметра *headers* в методі *get,* можна вказати особливі параметри за якими потрібно здійснювати HTTP з'єднання, це особливо корисно при багаторазовому парсингу одного джерела, адже тоді ми зможемо уникнути блокування .

Також в бібліотеці існує клас *requests.Session(),* який дозволяє здійснювати під час однієї сесії декілька запитів одночасно.

3.3.2.2 BeautifulSoup4

BeautifulSoup4 [11] - це зручна бібліотека для роботи з файлами в HTML та XML форматах.

Спочатку оголошуємо клас *BeatifulSoup*:

soup = BeautifulSoup(response.text, "html.parser")

Тобто передаємо в клас *BeautifulSoup* текст з веб-сторінки, яку ми знайшли за запитом з *response* та обробляємо за допомогою HTML-парсера. Далі для прикладу ми можемо знайти всі *div*-теги з довільним класом, для цього потрібно ввести наступний код:

soup.findAll('a', class_=CLASS_NAME)

За таким принципом ми можемо доступатись до будь-яких тегів та інформації в них.

3.3.2.3 Selenium

Selenium [12] - бібліотека для автоматизації роботи веб-браузера. За допомогою цієї бібліотеки ми можемо відкривати інформацію котра не доступна при звичайному запиті на веб-сторінку за допомогою *requests*. Наприклад, ми можемо натискати деякі кнопки на веб-сторінці, які відкривають більше інформації, а згодом спарсити з уже відкритої сторінки нову інформацію.

Щоб розпочати роботу з Selenium ми повинні встановити спеціальний веб-драйвер, це замінник нашого веб-браузера та інструмент в якому будуть відтворюватись всі дії, які ми запрограмували. Згодом потрібно прописувати усі дії, які повинен зробити наш скрипт, подібно, як це робиться в бібліотеці *requests*.

3.3.3 NLTK

NLTK [13] - це бібліотека котра використовується для лінгвістичної та статистичної обробки природних мов.

Серед функцій першої необхідності для задач природної обробки мови:

- Токенізацію (розбиття тексту на лексеми)
- Стемінг (скорочення слів до їх значущої частини)
- Лемматизація (процес групування слів з різним закінченням проте за однаковим значенням, як один елемент)
- Список слів, які не несуть ніякої інформації (через їх надмірне вживання)
- N-грами, збірка слів з тексту по n слів
- Переведення тексту у векторний формат за допомогою Bag of Words, word2vec, GloVe

Проте дана бібліотека має і певні мінуси, зокрема:

- Має повільну швидкість виконання
- Не надає вбудованих словників для роботи з українською мовою
- Не проводить семантичний аналіз

3.3.4 Tensorflow

Tensorflow [14] - бібліотека для алгоритмів глибокого навчання з відкритим кодом, котра використовується для усього спектру завдань штучного інтелекту.

Бібліотека має ряд функцій, що робить її надзвичайно зручною для тренування нейромережевих алгоритмів:

- Забезпечує автоматичне диференціювання, яке потрібне для вирішення оптимізаційних задач, та лежить в серці алгоритму поширення зворотної помилки [15].
- Окрім звичайного тренування на CPU (процесорі), розширення tensorflow-gpu дозволяє тренування алгоритмів на GPU (відеокартах) без написання додаткових скриптів.
- Надає широкий функціонал для препроцесингу даних

- Дозволяє використовувати змішану точність та містить вбудовані алгоритм квантизації нейромереж для пришвидшення часу та ресурсів для тренування нейромережевого алгоритму [16].
- Містить вбудовану архітектуру найбільш популярних нейронних алгоритмів.
- Надає функціонал для створення власних модулів. Наприклад, створення окремих шарів нейронної мережі, модулі для препроцесингу вхідних даних.

3.4 PostgreSQL

PostgreSQL [17] - реляційна система керування базами даних з відкритим кодом написана в 1996 році на мові програмування C.

PostgreSQL має багато переваг над своїми конкурентами, проте в той же час залишається безкоштовною. Автори PostgreSQL, а ним може стати будь-хто, зробили великий вклад в безпеку бази даних, як на рівні передавання інформації через різні безпечні протоколи, так і на рівні безпеки передавання транзакцій за властивостями ACID (*Atomicity - атомарність, Consistency - узгодженість, послідовність, Isolation - ізольованість, Durability - довговічність*).

Для роботи з текстовими даними PostgreSQL надає вбудований повний пошук по них, тобто знаходження певних слів або речень по певному спеціальному запиту. Також забезпечується функція *similiarity*, котра дозволяє визначити близькість двох текстів або слів (по написанню). Цікавим доповненням є можливість написання користувацьких функцій всередині системи керування базою даних, зокрема на мові програмування Python, що значно розширює можливості PostgreSQL.

Серед інших переваг PostgreSQL - масштабованість, підтримка багатьох видів даних, навіть у географічному форматі, а також можливість визначити свої власні типи даних та переписувати уже наявні функції.

3.5 Genesis Cloud

Очевидно, що телеграм-бот не може постійно функціонувати з локального пристрою. Це викликає ряд незручностей, та перешкоджає його стабільності. Тому прийнято завантажувати телеграм-ботів на певні віддалені хмарні сервіси, на яких вони працюють цілодобово.

Одним з нових на ринку хмарних технологій є сервіс Genesis Cloud [18]. Даний сервіс надає можливість комплектації власних серверів, зокрема з найкращими GPU серії Nvidia GeForce RTX 3090, 3080, що особливо корисно для навчання нейромережевих алгоритмів.

Підключення до таких серверів відбувається за допомогою генерованих SSH ключів, або що менш рекомендовано з точки зору безпеки, за допомогою пароля.

Оскільки програмний код даної бакалаврської роботи повністю підтримується системою контролю версій Git, то його завантаження та підтримування на сервері не викликає ніяких труднощів.

3.6 Нейромережеві алгоритми для завдань NLP

Для роботи з завданням природної обробки мови існує ряд алгоритмів машинного навчання, найпопулярніші з яких:

- Алгоритм Наївного Байєса
- Логістична регресія
- Дерево рішень

- Support Vector Machines

Такі алгоритми втрачають свою популярність, оскільки працюють лише з статистичними закономірностями, не вдаючись у семантику заданого тексту. На противагу такому підходу з'явились нейромережеві алгоритми. Завдяки більш комплексній структурі, такі алгоритми здатні краще вловлювати суть тексту, а отже і давати більш точні результати.

3.6.1 Нейромережевий алгоритм BERT

Одним з найкращих нейромережевих алгоритмів на даний момент є BERT (Bidirectional Encoder Representations from Transformers). Вперше статтю про BERT було опубліковано інженерами-дослідниками компанії Google у 2018 році [19].

В даній статті було запропоновано дві модифікації (див. рис.):

- Базова, має близько 110 мільйонів параметрів, та складається з 12 трансформер шарів, 12 шарів "уваги".
- Розширена, має близько 340 мільйонів параметрів, та складається з 24 трансформер шарів, 16 шарів "уваги".

Обидві модифікації тренувались на великих текстових корпусах, зокрема на книжковому корпусі який має більше 750 мільйонів слів, а також на додатковому корпусі котрий складається з статей Вікіпедії та має близько 2.5 мільярда слів.

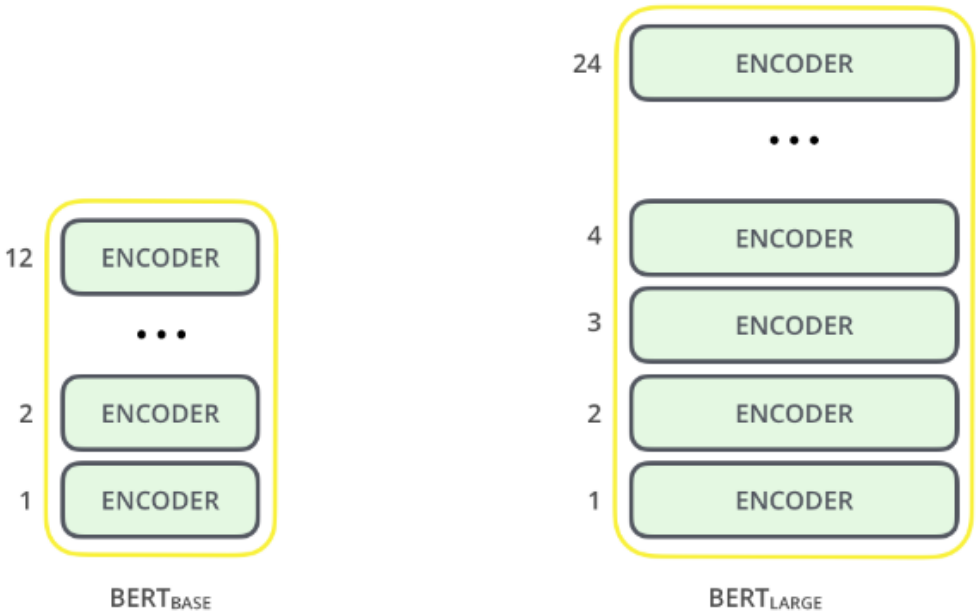


Рисунок 9 Архітектура BERT Base та BERT Large [20]

BERT показав свою ефективність в пошуковику Google для англійської мови, а згодом був імплементований і для інших мов [21]. Технології, засновані на BERT мають використання в задачах сумаризації, передбачення речень, класифікації настроїв та інших. Існують різні варіації моделі BERT передтренованої для різних задач, зокрема SciBERT – модель для наукових статей, BioBERT -модель для роботи з біомедичними текстами та інші.

Вхідний текст для алгоритму спочатку піддається токенізації на натренованому словнику, в базовій модифікації це близько 32 тисяч слів, якщо ж слово відсутнє в словнику воно ділиться на частини, які в ньому присутні. Згодом як ми бачимо на рисунку 10 токенізовані представлення, або їх ще називають ембедінги, змішуються зі своїми відповідними векторними та позиційними ембедінгами.

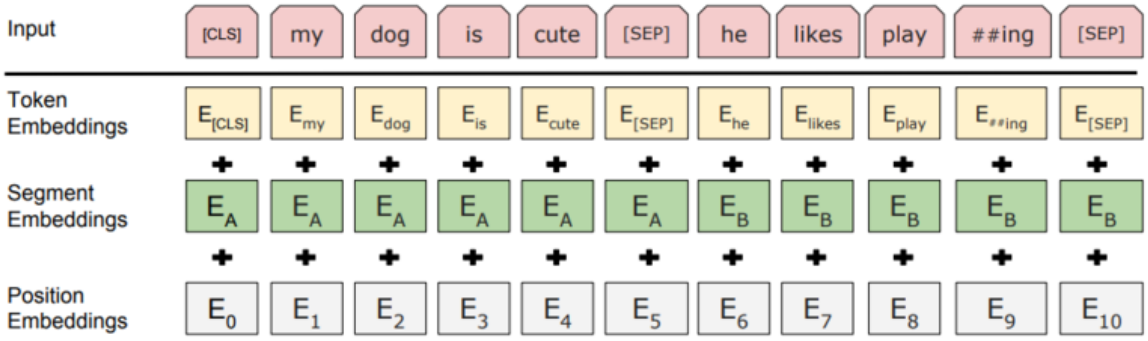


Рисунок 10 Перетворення вхідних даних для алгоритму BERT [19]

Оскільки модель BERT складається з сотень мільйонів параметрів зручно використовувати передтреновані раніше моделі та дотренувати їх для своєї локальної проблеми. При тренуванні на вхід BERT використовується описане вище представлення, проте деякі токени приховані (див. Рисунок 11). Завдяки цьому алгоритм навчається розуміти сенс речень тому, що виконує задачі передбачення наступного речення завдяки минулим, а також змістовно заповнює пропущений токен в реченні.

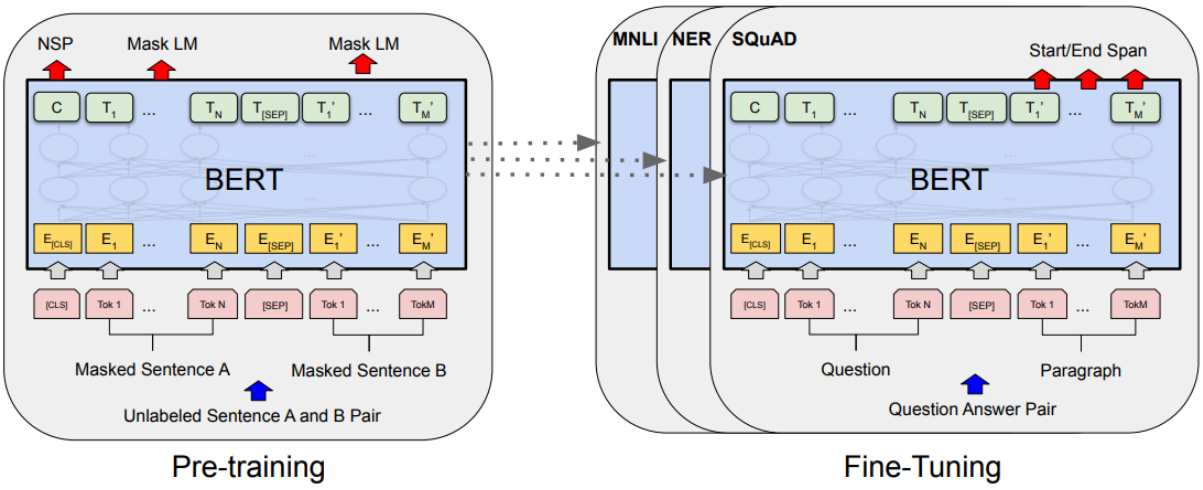


Рисунок 11 Передтренування та донавчання BERT [19]

Після донавчання одного з підвидів алгоритму BERT під назвою *uncased_L-12_H-768_A-12,* що означає модель, яка натренована незалежно від регістру тексту, та має 12 шарів “уваги”, 12 трансформер шарів, а також 768 прихованих шарів, та має наступні гіперпараметри:

- Розмір одного поділу при тренуванні (batch) - 18

- Крок навчання – $3 * 10^{-6}$
- Кількість епох - 3
- Максимальна кількість токенів на вході - 75
- Відсоток даних з вибірки, які будуть витрачені на те щоб зробити початкове “прогрівання” (під “прогріванням” розуміється процес зниження кроку навчання для уникнення перенавчання) - 0.15

В роботі використовувалась одна з широкопоширених реалізації BERT [24] котра в поєднанні з описаними вище параметрами дала 90.2 % точності.

3.6.2 Нейромережевий алгоритм на основі LSTM

Оскільки BERT оперує мільйонами параметрів, то на етапі передбачення даний алгоритм займає значно більший час, аніж алгоритми з меншою кількістю шарів. Тому в боті було реалізовано два класифікаційні алгоритми:

- Алгоритм BERT, який є точним проте повільним (для прикладу в соціальній мережі Twitter отримує результат до 1 хвилини)

- Алгоритм на основі шарів LSTM, котрий має меншу точність проте працює швидше (для прикладу в соціальній мережі Twitter отримує результат до 20 секунд)

На початку роботи користувач може обрати, який саме він бажає використовувати для передбачень.

LSTM – це нейромережевий рекурентний алгоритм, який є зручним для роботи з текстом, оскільки як і BERT навчається передбачати наступні токени на основі попередніх, проте у більш примітивній формі.

Створена нейронна мережа має наступну структуру:

Layer (type)	Output Shape	Param #
embedding_7 (Embedding)	(None, 6410, 48)	444000
bidirectional_14 (Bidirectio	(None, 6410, 128)	57856
bidirectional_15 (Bidirectio	(None, 6410, 64)	41216
global_max_pooling1d_8 (Glob	(None, 64)	0
dense_26 (Dense)	(None, 256)	16640
dropout_20 (Dropout)	(None, 256)	0
dense_27 (Dense)	(None, 128)	32896
dropout_21 (Dropout)	(None, 128)	0
dense_28 (Dense)	(None, 64)	8256
dropout_22 (Dropout)	(None, 64)	0
dense_29 (Dense)	(None, 5)	325

Total params: 601,189
Trainable params: 601,189
Non-trainable params: 0

Рисунок 12 Схема “швидкого” класифікаційного алгоритму

Як ми бачимо дана нейронна мережа на свій вхід бере ембеденгові представлення, згодом це представлення йде на вхід двох двонаправлених шарів LSTM. Після цього використовується Max Pooling шар, який дозволяє згладити чутливість до даних, та за рахунок цього створити модель, котра здатна узагальнювати наші дані. Також для усунення проблеми перенавчання між лінійними шарами застосовуються Dropout шари, котрі з заданою ймовірністю викидають нейрони в процесі навчання.

Даний нейронний алгоритм тренувався на 15 епохах, та має такі показники по метриці точності (accuracy):

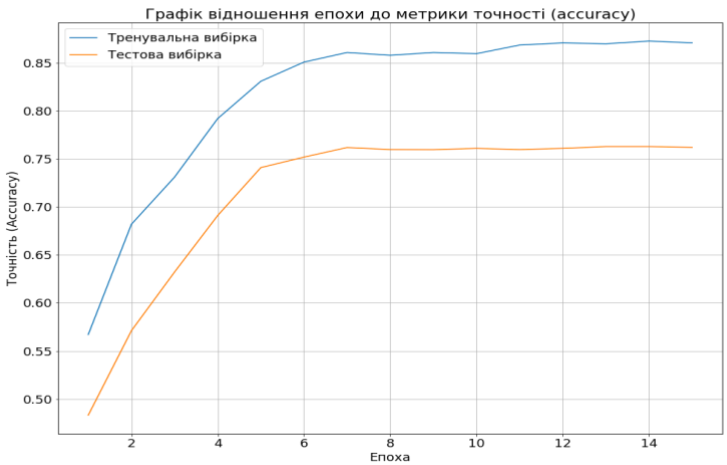


Рисунок 12 Графік метрики точності залежно від епохи тренування

Під час тренування використовувалась функція втрат - категоріальна кросентропія. Графік залежності цієї функції від епохи зображено на рисунку 13:

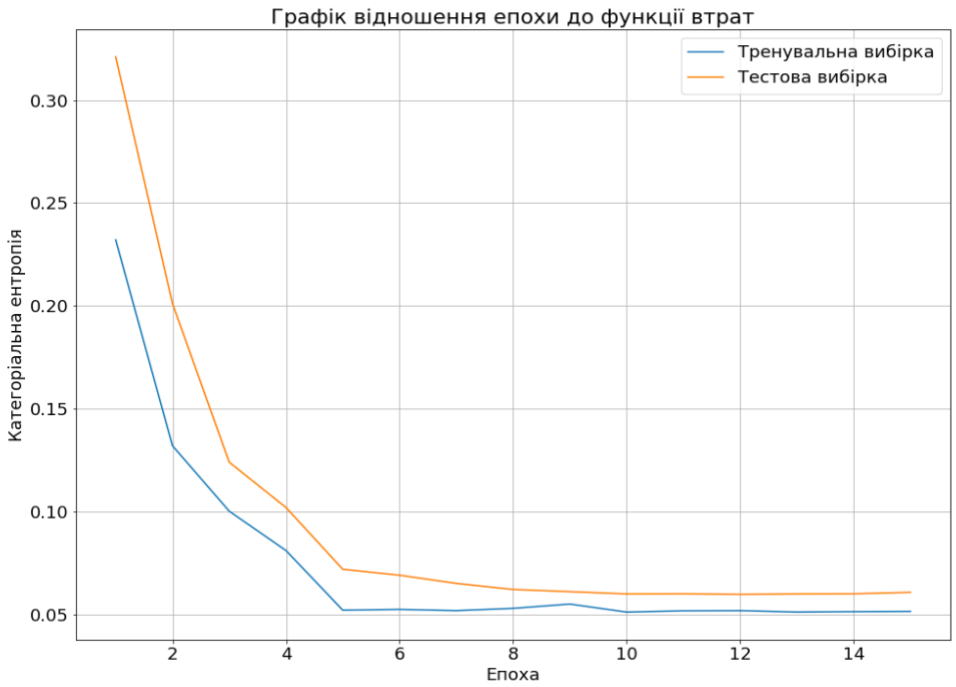


Рисунок 13 Залежність функції втрат від епохи

Як ми бачимо з рисунків 12, 13 після 8 епохи починається перенавчання моделі, що є “запам’ятовуванням” тренувальної вибірки. Тому щоб зберегти здатність алгоритму узагальнювати вирішення задачі незалежно від вхідних даних в телеграм-боті ми використали ваги нейронної мережі на 8 епосі.

ВИСНОВКИ

В теоретичній частині кваліфікаційної роботи було досліджено та описано розробку телеграм-бота, методики парсингу інформації з веб-сайтів, а також нейромережеві алгоритми для завдань обробки запитів природної мови.

У першому розділі було описано предмет та середовище розробки, оцінено наявні аналоги телеграм-ботів зі схожим функціоналом.

У другому розділі представлено функціональний аналіз розробленого програмного продукту. Складено відповідні схеми бази даних та міжмодульної структури.

Третій розділ містить інформацію про інструментальні засоби, які використовувались для розробки телеграм-бота, включно з алгоритмами класифікації та парсингу.

В результаті виконання практичної частини кваліфікаційної роботи було розроблено телеграм-бота для парсингу інформації з соціальних мереж, зокрема Google, Instagram, Reddit, Twitter та Pinterest за певним користувацьким запитом. Також даний програмний продукт містить в собі алгоритми класифікації, які застосовуються до зібраної інформації, та надає тексту одну з категорій – Політика, Розваги, Спорт, Техніка або ж Бізнес.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Статистика кількості користувачів месенджеру Телеграм. Станом на 12 січня 2021 року. Особистий канал Павла Дурова [Електронний ресурс] https://t.me/durov/147
2. Приватність месенджеру Телеграм. Офіційний сайт Telegram [Електронний ресурс] https://telegram.org/faq#q-how-secure-is-telegram
3. Офіційний телеграм-бот Укрзалізниці [Електронний ресурс] https://t.me/Ukrzaliznytsia_Tickets_Bot
4. Офіційний телеграм-бот monobank [Електронний ресурс] https://t.me/monobankbot
5. Можливості телеграм-ботів. Офіційний сайт Telegram [Електронний ресурс] https://core.telegram.org/bots#1-what-can-i-do-with-bots
6. Телеграм-бот для завантаження музики з YouTube Music - LyBot [Електронний ресурс] https://t.me/LyBot
7. Телеграм-бот для пошуку посилань - GoogleBot [Електронний ресурс] https://t.me/GoogleDEBot
8. Телеграм-бот для відслідковування в Instagram [Електронний ресурс] https://t.me/allsaverbot
9. Бібліотека telebot [Електронний ресурс] https://github.com/eternnoir/pyTelegramBotAPI
10. Бібліотека requests [Електронний ресурс] https://docs.python-requests.org/en/latest/

11. Бібліотека BeatifulSoup4 [Електронний ресурс] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

12. Бібліотека Selenium [Електронний ресурс] https://selenium-python.readthedocs.io/

13. Бібліотека NLTK [Електронний ресурс] https://www.nltk.org/

14. Бібліотека Tensorflow [Електронний ресурс] https://www.tensorflow.org/

15. David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams. Learning representations by back-propagating errors, 1986

16. Автоматичне тренування зі змішаною точністю [Електронний ресурс] https://www.tensorflow.org/guide/mixed_precision

17. Офіційний сайт системи керування базами даних - PostgreSQL [Електронний ресурс] https://www.postgresql.org/

18. Офіційний сайт сервісу Genesis Cloud [Електронний ресурс] https://www.genesiscloud.com/

19. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, 2018.

20. BERT base та BERT large [Електронний ресурс] https://iq.opengenus.org/bert-base-vs-bert-large/

21. Використання BERT в Google [Електронний ресурс]

https://www.youtube.com/watch?v=2lR8Fzays4I

22. BERT language model [Електронний ресурс] https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model
23. BERT/GPT [Електронний ресурс] https://www.researchgate.net/figure/The-Difference-Between-BERT-and-Open-GPT-extracted-from-Devlin-et-al-2-Figure-1_fig1_334413801
24. BERT code [Електронний ресурс] https://github.com/google-research/bert

ДОДАТОК А

Демонстрація програмного продукту

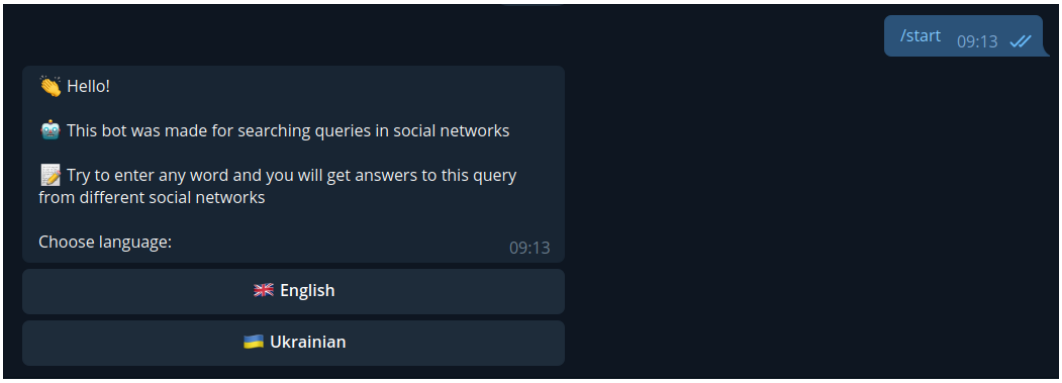


Рисунок 14 Початок роботи телеграм-бота

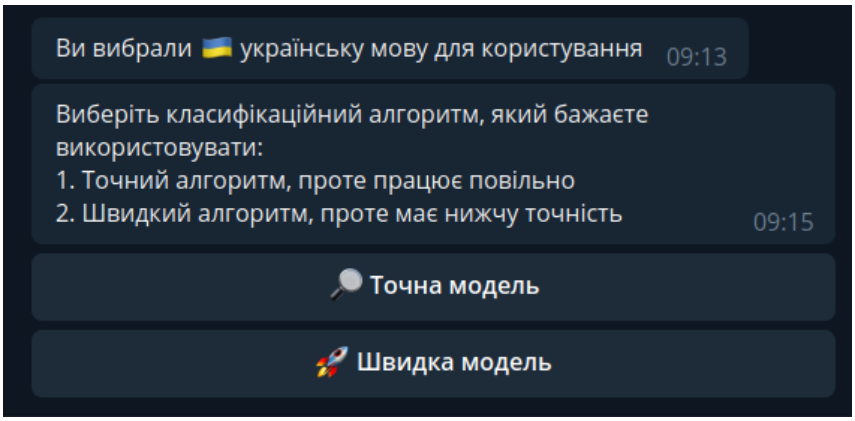


Рисунок 15 Вибір класифікаційного алгоритму

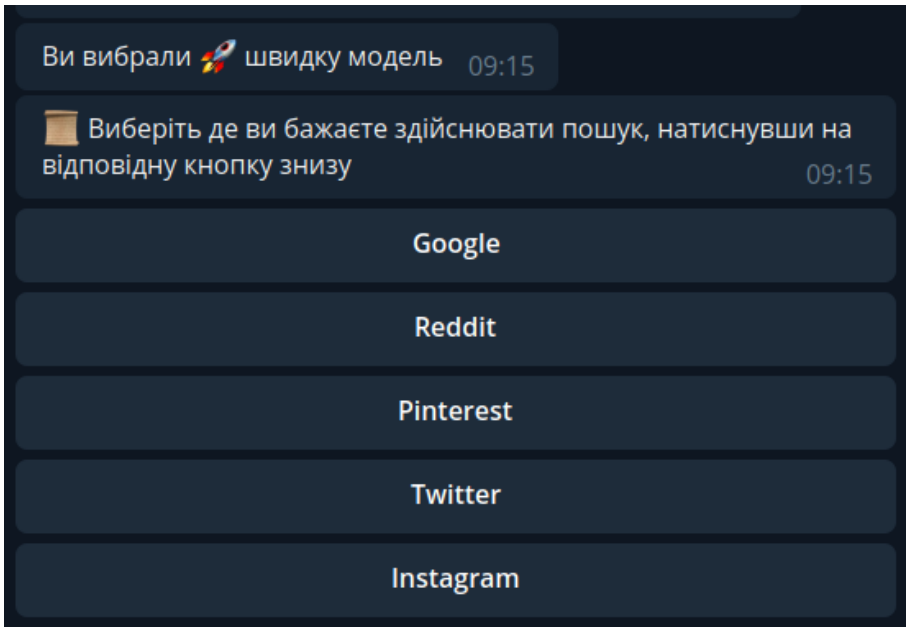


Рисунок 16 Вибір соціальної мережі для пошуку

Рисунок 17 Вибір Google, як соціальної мережі

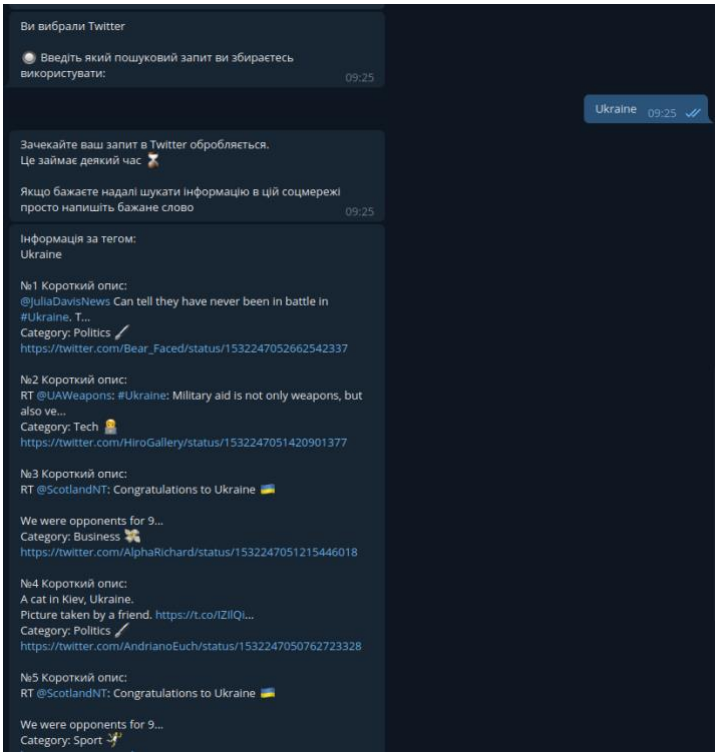


Рисунок 18 Вибір Twitter, як соціальної мережі

Рисунок 19 Натиснуто кнопку вивести ще 10 посилань

ДОДАТОК Б

Програмний код

Клас для парсингу Google:

```
class GoogleParser:
        """
        Клас для парсингу посилань з Google
        """

        def __init__(self, database, model, user_id: int) -> None:
        self.database = database
        self.model = model
        (
        self.query_word,
        self.number,
        _,
        self.language,
        ) = self.database.select_socialnetwork_info(user_id=user_id)
        self.number = 1 + 10 * (self.number - 1)
        self.shorter_url = pyshorteners.Shortener()

        @staticmethod
        def parse_page(url):
        """
        Препроцесинг HTML дерева
        """
        try:
        response = requests.get(url, timeout=3)
        except:
        return [""], response.status_code
        text = response.text
        text = re.sub(r"\n", " ", text)
        text = re.sub(r"\t", " ", text)
        text = re.sub(re.compile(r"<script.*?<\/script>"), "", text)
        text = re.sub(re.compile(r"<style.*?<\/style>"), "", text)
        text = re.sub(re.compile("<.*?>", re.MULTILINE), "", text)
        text = re.sub(r"\n|\s{2,}", " ", text)
        return [
        "".join(re.findall("[a-zA-Z0-9_ ]", text.lower()))
        ], response.status_code

        def link_title(self, url: str) -> Tuple[str, int]:
        """
        Функція витягує назву сайту через посилання на сайт
        """
        response = requests.get(url, timeout=5)
```

```
        content = response.content
        try:
        tree = fromstring(content.decode())
        except:
        tree = fromstring(content)
        last = tree.findtext(".//title")

        return self.title_for_error(url) if not last else last, response.status_code

        def title_for_error(self, link: str) -> str:
        """
        Оскільки не кожен сайт дозволяє парсити свою назву, то коли фукнція link_title
повертає
        невдалий статус код, тоді ми дістаємо назву сайту вручну з посилання
        """
        splited_domain = link.split(".")

        if "www" in splited_domain[0]:
        title = splited_domain[1]
        else:
        title = re.findall(r'https?://[\'"]?([^\'" >]+)', splited_domain[0])
        title = self.query_word if not title else title[0]

        return title.title()

        @staticmethod
        def row_template(
        language: str,
        previous_rows: str,
        current_number: int,
        title: str,
        link: str,
        category: str,
        ) -> str:
        """
        Форматування стрічки в кінцевому тексті
        """
        return f"\n{previous_rows}№{current_number} {'Заголовок:' if language == 'ukrainian'
else 'Header:'}\n{title}\n{'Категорія:' if language == 'ukrainian' else 'Category:'}
{category}\n{link}\n"

        def search(self) -> str:
        """
        Виведення посилань по пошуковому запиту в Google
        """

        result_text = (
        f"Інформація за запитом:\n{self.query_word}\n\n"
```

```
if self.language == "ukrainian"
else f"Information about the query:\n{self.query_word}\n\n"
)

for link in search(
self.query_word, tld="co.in", num=10, start=self.number, stop=10, pause=2.0
):
try:
        title, title_status_code = self.link_title(link)

        if title_status_code != 200:  # відсіюємо невдалі запити

        title = self.title_for_error(link)
        else:
        # пробіл або більше чим 1 відступ з назви видаляється
        title = re.sub(r"\n|\s{2,}", "", title)

        preprocessed_text, code = self.parse_page(link)

        if code != 200:
        category = (
        "Парсинг сайту не дозволений"
        if self.language == "ukrainian"
        else "Parsing of this site is not allowed"
        )
        else:
        category = self.model.predict(preprocessed_text)

        result_text = self.row_template(
        self.language,
        result_text,
        self.number,
        title,
        self.shorter_url.tinyurl.short(link),
        category,
        )

        self.number += 1
except:
        title = self.title_for_error(link)
        result_text = self.row_template(
        self.language,
        result_text,
        self.number,
        title,
        self.shorter_url.tinyurl.short(link),
        (
        "Парсинг сайту не дозволений"
```

```
                if self.language == "ukrainian"
                else "Parsing of this site is not allowed"
                ),
                )
                self.number += 1

        return (
        (
                f"😢 Сформуйте будь ласка запит іншим чином.\nНа запит
{self.query_word} не знайдено жодного посилання."
                if self.language == "ukrainian"
                else f"😢 Please formulate the query differently.\nNo link was found for query
{self.query_word}"
        )
        if not result_text
        else result_text
        )
```

Клас для парсингу Reddit:

```
class RedditParser:
        def __init__(self, database, model, user_id: int) -> None:
        self.database = database
        self.model = model
        (
        self.query_word,
        self.number,
        _,
        self.language,
        ) = self.database.select_socialnetwork_info(user_id=user_id)
        self.number = 1 + 10 * (self.number - 1)
        self.reddit_client = praw.Reddit(
        client_id=config.REDDIT_CLIENT_ID,
        client_secret=config.REDDIT_CLIENT_SECRET,
        user_agent=config.REDDIT_USER_AGENT,
        username=config.REDDIT_USERNAME,
        password=config.REDDIT_PASSWORD,
        )

        def post_link(self, submission: str) -> str:
        """
        Посилання на пост в сабредіті
        """
        return f"https://www.reddit.com/r/{self.query_word}/comments/{submission}"

        def search(self) -> str:
        """
        Метод для виведення тексту з заданого сабредіту
```

```
        """

        link = ""

        try:
        subreddit = self.reddit_client.subreddit(self.query_word)
        for number, submission in enumerate(
                subreddit.top(limit=self.number * 10 + 1)
        ):

                if number < self.number:
                continue

                link += "\n" + (
                "№{} Назва посту: \n{}\nКатегорія: {}\n{}\n "
                if self.language == "ukrainian"
                else "№{} Title of the post: \n{}\nCategory: {}\n{}\n"
                ).format(
                number,
                submission.title,
                self.model.predict([f"{submission.title} {submission.selftext}"]),
                self.post_link(submission),
                )

                if number == self.number + 9:
                return (
                "Інформація з сабредіту:\n{}\n{}"
                if self.language == "ukrainian"
                else "Information from subdreddit:\n{}\n{}"
                ).format(self.query_word, link)

        except:
        return (
                "Назви такого сабредіту немає"
                if self.language == "ukrainian"
                else "There is no name for such a subreddit"
        )

        return (
        (
                "Назви такого сабредіту немає або інформація в даному сабредіті
закінчилась"
                if self.language == "ukrainian"
                else "There is no name for such a subreddit or the information in this sabredit
is over"
        )
        if not link
        else (
```

```
                (
                "Інформація з сабредіту:\n{}\n{}"
                if self.language == "ukrainian"
                else "Information from subdreddit:\n{}\n{}"
                ).format(self.query_word, link)
                + (
                "😢 Вибачте, посилання закінчились"
                if self.language == "ukrainian"
                else "😢 Sorry, the links has expired"
                )
        )
        )
```

Клас для парсингу Twitter:

```
class TwitterParser:
        def __init__(self, database, model, user_id: int) -> None:
        self.database = database
        self.model = model
        (
        self.query_word,
        self.number,
        _,
        self.language,
        ) = self.database.select_socialnetwork_info(user_id=user_id)
        self.number = 1 + 10 * (self.number - 1)
        self.twitter_client = TwitterSearch(
        consumer_key=config.TWITTER_API_KEY,
        consumer_secret=config.TWITTER_API_SECRET_KEY,
        access_token=config.TWITTER_API_TOKEN,
        access_token_secret=config.TWITTER_API_TOKEN_SECRET,
        )

        @staticmethod
        def preproccess(text: str) -> List[str]:
        """
        Базовий препроцесинг повідомлення
        """
        text = re.sub(r"\n", " ", text)
        text = re.sub(r"\t", " ", text)
        text = re.sub(r"\n|\s{2,}", " ", text)
        return ["".join(re.findall("[a-zA-Z0-9_ ]", text.lower()))]

        def search(self) -> str:
        """
        Метод для виведення тексту за певним тегом в Twitter
        """
```

```
        tso = TwitterSearchOrder()
        tso.set_keywords([self.query_word])
        tso.set_include_entities(False)

        link = ""

        try:
        posts = self.twitter_client.search_tweets_iterable(tso)
        for number, tweet in enumerate(posts):

                if number < self.number:
                continue

                link += (
                "\n№{} Короткий опис:\n{}\nCategory: {}\n{}\n"
                if self.language == "ukrainian"
                else "\n№{} Brief overview:\n{}\nCategory: {}\n{}\n"
                ).format(
                number,
                tweet["text"][:70] + "...",
                self.model.predict(self.preproccess(tweet["text"])),
                f"https://twitter.com/{tweet['user']['screen_name']}/status/{tweet['id']}",
                )

                if number == self.number + 9:
                return (
                "Інформація за тегом:\n{}\n{}"
                if self.language == "ukrainian"
                else "Information from tag:\n{}\n{}"
                ).format(self.query_word, link)

        except:
        return (
                "Інформації за таким тегом немає"
                if self.language == "ukrainian"
                else "There is no info for this tag"
        )

        return (
        (
                "Інформації за даним тегом немає або інформація за даним тегом
закінчилась"
                if self.language == "ukrainian"
                else "There is no info for this tag or the information in this tag is over"
        )
        if not link
        else (
                (
```

```
                "Інформація за тегом:\n{}\n{}\n"
                if self.language == "ukrainian"
                else "Information from tag:\n{}\n{}\n"
                ).format(self.query_word, link)
                + (
                "😢 Вибачте, посилання закінчились"
                if self.language == "ukrainian"
                else "😢 Sorry, the links has expired"
                )
        )
        )
```

Класи для парсингу Instagram, Reddit мають схожу з наведеними вище структуру.

Клас для роботи з базою даних:

```
class Database:
        """
        Клас для роботи з базою даних за допомогою СКБД PostgreSQL
        """

        def __init__(self) -> None:
        """
        Створення або під'єднання до існуючої бази даних tg_bot
        """
        self.parameters = {
        "database": config.POSTGRES_DATABASE,
        "user": config.POSTGRES_USER,
        "password": config.POSTGRES_PASSWORD,
        "host": config.POSTGRES_HOST,
        "port": config.POSTGRES_PORT,
        }
        self.database_name = "tg_bot"
        self._init_db()
        self.connection = psycopg2.connect(**self.parameters)
        self._create_tables()

        def _init_db(self) -> None:
        """
        Ініціалізація бази даних
        """
        connection = psycopg2.connect(**{**self.parameters, "database": None})
        cursor = connection.cursor()
        connection.autocommit = True
        cursor.execute(
        "SELECT 1 FROM pg_catalog.pg_database WHERE datname = %s",
        (self.database_name,),
        )
        if not cursor.fetchone():
```

```
        cursor.execute(f"CREATE DATABASE {self.database_name}")
        print(f"БАЗУ ДАНИХ <{self.database_name}> було створено")
        else:
        print(f"БАЗА ДАНИХ <{self.database_name}> вже існує")
        cursor.close()

        def _create_tables(self) -> None:
        """
        Створення таблиць в базу даних tg_bot
        """
        create_telegramuser_table = """CREATE TABLE IF NOT EXISTS telegramuser(
        user_id INT NOT NULL,
        firstname VARCHAR(255),
        lastname VARCHAR(255),
        username VARCHAR(255),
        PRIMARY KEY(user_id)
        )"""
        create_socialnetwork_table = """CREATE TABLE IF NOT EXISTS socialnetwork(
        user_id INT NOT NULL,
        query_word VARCHAR(255),
        scroll_number INT,
        social_network VARCHAR(255),
        language VARCHAR(255),
        model_type INT,
        PRIMARY KEY(user_id),
        FOREIGN KEY(user_id) REFERENCES telegramuser(user_id)
        ON DELETE CASCADE
        )"""
        cursor = self.connection.cursor()
        cursor.execute(create_telegramuser_table)
        cursor.execute(create_socialnetwork_table)
        self.connection.commit()
        cursor.close()

        def insert_info(
        self, table_name: str, row_name: str, row_data: str, user_id: int
        ) -> None:
        """
        Вставляє або оновлює інформацією в певній таблиці, та в певній колонці
        """
        check_data = f"""SELECT 1 FROM {table_name} WHERE user_id = %s"""
        cursor = self.connection.cursor()
        cursor.execute(check_data, (user_id,))
        if cursor.fetchone():
        update_data = f"UPDATE {table_name} SET {row_name} = %s WHERE user_id =
%s"
        cursor.execute(update_data, (row_data, user_id))
        else:
```

```
        inser_data = (
                f"INSERT INTO {table_name} (user_id, {row_name}) VALUES (%s, %s)"
        )
        cursor.execute(inser_data, (user_id, row_data))
        self.connection.commit()
        cursor.close()

        def insert_tginfo(
        self,
        user_id: int,
        firstname: str = None,
        lastname: str = None,
        username: str = None,
        ) -> None:
        """
        Вставляє в таблицю telegramuser інформацію про ім'я, прізвище
        та тег користувача
        """
        insert_info = "INSERT INTO telegramuser (user_id, firstname, lastname, username)
VALUES (%s, %s, %s, %s)"
        cursor = self.connection.cursor()
        check_data = """SELECT 1 FROM telegramuser WHERE user_id = %s"""
        cursor.execute(check_data, (user_id,))
        if not cursor.fetchone():
        cursor.execute(insert_info, (user_id, firstname, lastname, username))
        self.connection.commit()
        cursor.close()

        def select_info(
        self, table_name: str, row_name: str, user_id: int
        ) -> Tuple[str, int]:
        """
        Вибірка інформація з певної таблиці, за певною колонкою
        """
        select_data = f"SELECT {row_name} FROM {table_name} WHERE user_id = %s"
        cursor = self.connection.cursor()
        cursor.execute(select_data, (user_id,))
        query_output = cursor.fetchone()
        self.connection.commit()
        cursor.close()
        return query_output[0] if query_output else None

        def select_socialnetwork_info(self, user_id: int) -> Tuple[str, int]:
        cursor = self.connection.cursor()
        select_data = "SELECT query_word, scroll_number, social_network, language
FROM socialnetwork WHERE user_id = %s"
        cursor.execute(select_data, (user_id,))
        query_output = cursor.fetchone()
```

```
        self.connection.commit()
        cursor.close()
        return query_output

        def delete_info(self, user_id: int) -> None:
        """
        Видаляє користувача з бази даних.
        Оскільки таблиця socialnetwork має зовнішній ключ до таблиці telegramuser
        з параметром ON DELETE CASCADE, то при видаленні з таблиці telegramuser
        користувач автоматично видалиться і з socialnetwork
        """
        delete_socialnetwork_table = "DELETE FROM telegramuser WHERE user_id = %s"
        cursor = self.connection.cursor()
        cursor.execute(delete_socialnetwork_table, (user_id,))
        self.connection.commit()
        cursor.close()

        def _version(self) -> str:
        """
        В бакалаврській роботі використовувалась версія PostgreSQL 13.4
        """
        cursor = self.connection.cursor()
        cursor.execute("SELECT version()")
        db_version = cursor.fetchone()
        cursor.close()
        return db_version
```

Приклад з модуля, який відповідає за двомовність

```
class Language:
        start = {
        "ukrainian": """
👏 Привіт!

👹 Цей бот створений для запитів з різних соцмереж за ключовим словом

📝 Спробуй ввести будь-яке слово  і ви отримаєте відповіді на цей запит із різних соцмереж

""",
        "english": """
👏 Hello!

👹 This bot was made for searching queries in social networks

📝 Try to enter any word and you will get answers to this query from different social networks

""",
```

```
}
```

Алгоритм на базі LSTM:

```
class LSTMClassifier:
    """
    Клас котрий містить класифікатор заснований на
    двонапревленій нейромережі LSTM
    """

    def __init__(
    self,
    weights_id: Optional[str] = "1us-TL81TTSycfo-rakCqM8-DnqMlbD3F",
    weights_path: Optional[str] = "lstm_weights.h5",
    ) -> None:
    """
    На етапі ініціалізації з Google Drive завантажується архів
    archive.zip
    - train.csv
    - lstm_weights.h5
    Ініціалізується токенайзер та модель з завантаженими вагами
    """

    # Завантажуємо з Google Drive архів описаний в docstring
    if weights_id:
    archive_name = "archive.zip"
    gdown.download(
            id=weights_id,
            output=archive_name,
            quiet=False,
    )
    with zipfile.ZipFile(archive_name, "r") as zip_file:
            zip_file.extractall()

    self.voc_size = 9250
    self.emb_size = 48
    self.max_len = 6410
    self.graph = tf.get_default_graph()
    self.model = self._model_architecture()
    self.model.load_weights(weights_path)
    self._labels = [
    "Business 💸",
    "Entertainment 👪",
    "Politics 🖉",
    "Tech 👨💻",
    "Sport 🤸",
    ]
    self.tok = Tokenizer(num_words=self.voc_size)
    train_df = pd.read_csv("train.csv", index_col=0)
```

```
        self.tok.fit_on_texts(train_df["text"].values)

        def predict(self, text: List[str]):
        """
        Метод для виведення категорії по заданому тексту
        """
        with self.graph.as_default():
        init = tf.global_variables_initializer()
        get_session().run(init)
        return [
                self._labels[np.argmax(i)]
                for i in self.model.predict(self._preprocessing(text))
        ][0]

        def _preprocessing(self, text: List[str]):
        """
        Препроцесинг тексту, зокрема за допомогою токенізації
        на натренованому раніше токенізаторі
        """
        test_seq = pad_sequences(self.tok.texts_to_sequences(text), maxlen=self.max_len)
        return test_seq

        def _model_architecture(self):
        """
        Архітектура нейромережевої моделі
        """
        model = Sequential()
        model.add(
        Embedding(self.voc_size, self.emb_size, input_length=self.max_len)
        )
        model.add(Bidirectional(LSTM(64, return_sequences=True)))
        model.add(Bidirectional(LSTM(32, return_sequences=True)))
        model.add(GlobalMaxPooling1D())
        model.add(Dense(256))
        model.add(Dropout(0.25))
        model.add(Dense(128))
        model.add(Dropout(0.25))
        model.add(Dense(64))
        model.add(Dropout(0.2))
        # 5 категорій
        model.add(Dense(5, activation="softmax"))
        return model
```

Алгоритм BERT:

```
import os
import zipfile
```

```
import numpy as np
import tensorflow as tf
import wget
import gdown

if not os.path.isfile(
    os.path.join(os.getcwd(), "model", "uncased_L-12_H-768_A-12.zip")
):
    wget.download(
        "https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-
12.zip",
        os.path.join(os.getcwd(), "model"),
    )

if not os.path.isfile("bert_helpers.zip"):
    gdown.download(
        id="1ivpuz_CpEXkXrelwXvHeWpp-eMgwqtwP", output="bert_helpers.zip", quiet=True
    )

with zipfile.ZipFile("bert_helpers.zip", "r") as zip_ref:
    zip_ref.extractall(os.path.join(os.getcwd(), "model"))

from model import modeling, run_classifier, tokenization


class Config:
    LEARNING_RATE = 3 * 1e-6
    TRAIN_EPOCHS = 5.0
    TRAIN_BATCH = 18
    WARMUP_PERCENT = 0.15
    MAX_SEQ_LENGTH = 75
    N_ITERATIONS = 8000
    N_TPU_CORES = 6
    N_CHECKPOINTS = 9250
    N_TRAIN_DATASET = 61911
    LOWER_CASE = True
    MODEL_FOLDER = "model_folder"
    OUTPUT_DIR = os.path.join(MODEL_FOLDER, "outputs")
    BERT_PRETRAINED_DIR = os.path.join(MODEL_FOLDER, "uncased_L-12_H-768_A-
12")
    VOC_FILE = os.path.join(BERT_PRETRAINED_DIR, "vocab.txt")
    CONF_FILE = os.path.join(BERT_PRETRAINED_DIR, "bert_config.json")
    INIT_WEIGHTS = os.path.join("model", "model_folder", "outputs-2", "model.ckpt-1237")


class BertClassifier(Config):
    def __init__(self) -> None:
        self._unzip_pretrained_bert()
```

```
    self._labels = [
        "Business 💸",
        "Entertainment 🎞",
        "Politics 🗞",
        "Sport 🤸",
        "Tech 🧑💻",
    ]

    self._label_list = [str(num) for num in range(len(self._labels))]
    self._tokenizer = tokenization.FullTokenizer(
        vocab_file=self.VOC_FILE, do_lower_case=self.LOWER_CASE
    )

    run_bert_config = tf.contrib.tpu.RunConfig(
        cluster=None,
        model_dir=self.OUTPUT_DIR,
        save_checkpoints_steps=self.N_CHECKPOINTS,
        tpu_config=tf.contrib.tpu.TPUConfig(
            iterations_per_loop=self.N_ITERATIONS,
            num_shards=self.N_TPU_CORES,
            per_host_input_for_training=tf.contrib.tpu.InputPipelineConfig.PER_HOST_V2,
        ),
    )

    n_train_steps = int(
        self.N_TRAIN_DATASET / self.TRAIN_BATCH * self.TRAIN_EPOCHS
    )
    n_warmup_steps = int(n_steps * self.WARMUP_PERCENT)

    model_fn = run_classifier.model_fn_builder(
        bert_config=modeling.BertConfig.from_json_file(self.CONF_FILE),
        num_labels=len(self._label_list),
        init_checkpoint=self.INIT_WEIGHTS,
        learning_rate=self.LEARNING_RATE,
        use_tpu=False,
        use_one_hot_embeddings=True,
        num_train_steps=n_train_steps,
        num_warmup_steps=n_warmup_steps,
    )

    self._estimator = tf.contrib.tpu.TPUEstimator(
        model_fn=model_fn,
        config= run_bert_config,
        use_tpu=False,
    )

def input_fn_builder(self, features, seq_length, drop_remainder):
```

```
    all_input_ids, all_input_mask, all_segment_ids, all_label_ids = [], [], [], []

    for feature in features:
        all_input_ids.append(feature.input_ids)
        all_input_mask.append(feature.input_mask)
        all_segment_ids.append(feature.segment_ids)
        all_label_ids.append(feature.label_id)

    def input_fn(params):

        num_examples = len(features)
        const = lambda type_list: tf.constant(
            type_list, shape=[num_examples, seq_length], dtype=tf.int32
        )
        example = tf.data.Dataset.from_tensor_slices(
            {
                "input_ids": const(all_input_ids),
                "input_mask": const(all_input_mask),
                "segment_ids": const(all_segment_ids),
                "label_ids": tf.constant(
                    all_label_ids, shape=[num_examples], dtype=tf.int32
                ),
            }
        )

        return example.batch(batch_size=100, drop_remainder=drop_remainder)

    return input_fn

def predict(self, text: str) -> str:
    """
    Метод для виведення категорії по заданому тексту
    """

    predict_features = run_classifier.convert_examples_to_features(
        self.create_example(np.array(text)),
        self._label_list,
        self.MAX_SEQ_LENGTH,
        self._tokenizer,
    )

    predict_input_fn = self.input_fn_builder(
        features=predict_features,
        drop_remainder=False,
        seq_length=self.MAX_SEQ_LENGTH,
    )
```

```
    result = self._estimator.predict(input_fn=predict_input_fn)
    preds = []
    for prediction in result:
        preds.append(np.argmax(prediction["probabilities"]))

    return self._labels[preds[0]]

def create_example(self, lines) -> list:
    return [
        run_classifier.InputExample(
            guid="test", text_a=line, text_b=None, label="0"
        )
        for line in lines
    ]

def _unzip_pretrained_bert(
    self, filename: str = "uncased_L-12_H-768_A-12.zip"
) -> None:
    """
    Розархівувати архів з BERT моделлю
    """

    if os.path.isdir(self.MODEL_FOLDER):
        return None

    with zipfile.ZipFile(
        os.path.join(os.getcwd(), "model", filename), "r"
    ) as zip_ref:
        zip_ref.extractall(self.MODEL_FOLDER)
```