НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»
спеціальності «121 Інженерія програмного забезпечення»

на тему: Алгоритми та ПЗ для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT

Виконав студент IV курсу, групи ІТ-91
(шифр групи)

Луцай Катерина Андріївна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Баклан І. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст.викл. Головченко М. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., доц., Резніков С. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії
Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 121 Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютерних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Луцай Катерина Андріївні
(прізвище, ім'я, по батькові)

1. Тема проєкту Алгоритм та ПЗ для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT

керівник проєкту Баклан Ігор Всеволодович, доцент, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2023 р. №2101-с
2. Термін подання студентом проєкту « 17 » червня 2023 року
3. Вихідні дані до проєкту: технічне завдання
4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних.
3) Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв'язання.
4) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
5) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань
2) Схема структурна станів системи
3) Архітектура програмного забезпечення
4)
5)
6)

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	10.03.2023	
3	Постановка та формалізація задачі	18.03.2023	
4	Розробка інформаційного забезпечення	27.03.2023	
5	Алгоритмізація задачі	3.04.2023	
6	Обґрунтування вибору використаних технічних засобів	7.04.2023	
7	Розробка програмного забезпечення	4.05.2023	
8	Налагодження програми	10.05.2023	
9	Виконання графічних документів	15.05.2023	
10	Оформлення пояснювальної записки	29.05.2023	
11	Подання ДП на попередній захист	06.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Катерина ЛУЦАЙ
(підпис) (ініціали, прізвище)

Керівник _____ Ігор БАКЛАН
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 30 таблиць, 15 рисунків та 46 джерел – загалом 105 сторінок.

Дипломний проєкт присвячений аналізу великих масивів коротких текстових даних, таких як запису у соціальних мережах, для визначення місцезнаходження авторів

Мета цього проєкту – покращення точності прогнозування місцезнаходження за текстом та надання гнучкої методології для маркування датасетів іншим розробникам.

Об'єкт дослідження: нейронні мережі сімейства представлення двонаправленого кодера у трансформерах (BERT) для аналізу натуральної мови (NLP)

Предмет дослідження: система прогнозування місцезнаходження за текстом на базі даних Твіттера

У Розділі 1 розглянуто минулі роботи в предметній області дослідження нейронних мереж та алгоритмів машинного навчання для вирішення задачі регресії до чисельного представлення місцезнаходження автора. Описано вимоги до програмного забезпечення на базі аналізу минулих підходів з використанням BERT, та поставлено задачі даної роботи. Визначено основний функціонал та спеціальні вимоги до реалізації алгоритмів, архітектури, та допоміжних програмних засобів.

Розділ 2 присвячений моделюванню архітектури нейронної мережі та конструюванню багатозадачних алгоритмів машинного навчання для обрахунки функцій втрат нестандартного вихідного формату. Описано архітектуру ПЗ Розробника та ПЗ Користувача у вигляді серверної частини Телеграм боту для демонстрації роботи найкращої з навчених моделей. Розглянуто розділення на класи модульної архітектури на мові Python та розподіл функціоналу за відповідними утилітами. Запропонований підхід використовує нейронні мережі для обробки природної мови (NLP) для оцінки місцезнаходження у вигляді пар координат (довгота, широта) та моделей

сумісних двовимірних розподілів (GMM) з обмеженням вихідного параметра що відповідає за коефіцієнт сферичної матриці коваріації.

У Розділі 3 розглянуто минулі роботи в предметній області дослідження з точки зору загальноприйнятий метрик ефективності (точності) прогнозування місцезнаходження. Показники ефективності показують, що середня похибка становить менше 30 км на світовому рівні і менше 15 км на рівні США для моделей, навчених і оцінених на текстових змінних контенту твітів (текст) і контексті їх метаданих (користувач, місце). Описано мануальне тестування навчання, оцінки та прогнозування місцезнаходження в ролях Розробника та Користувача.

Нарешті, Розділ 4 присвячений лише розгортанню серверної частини Телеграм боту, оскільки ПЗ Розробника призначене до локального запуску.

Програмне забезпечення впроваджено на високопродуктивному кластері на базі ОС з ядром Unix без графічного інтерфейсу користувача.

Результати роботи пройшли апробацію на рівнві директора департаменту European Laboratory for Learning and Intelligent Systems (ELLIS) Christroph H. Lampert'a та подані на публікацію в Journal of Spatial Information Science (JOSIS).

КЛЮЧОВІ СЛОВА: ПРОГНОЗУВАННЯ МІСЦЕЗНАХОДЖЕННЯ, ТРАНСФОРМЕРИ, НАБІР ДАНИХ ТВІТЕР, МАШИННЕ НАВЧАННЯ, ЗАДАЧА РЕГРЕСІЇ, СУМІСНА МОДЕЛЬ ГАУСА, БАГАТОЗАДАЧНЕ НАВЧАННЯ, ТЕЛЕГРАМ БОТ, АНАЛІЗ ДАНИХ, ОБРОБКА НАТУРАЛЬНОЇ МОВИ.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 30 tables, 15 figures and 46 sources – in total 105 pages.

The purpose of the diploma project is providing software to analyze large amounts of short text data, such as social media posts, to predict the location of authors.

The goal of this project is to improve the accuracy of geolocation prediction from text and provide a flexible methodology for labelling datasets for other developers.

Object of research: neural networks of the Bidirectional Encoder Representation in Transformers (BERT) family for natural language processing (NLP)

Subject of research: a text-based location prediction system based on custom Twitter data

Section 1 reviews past work in the subject area of neural networks and machine learning algorithms for solving the problem of regression to a numerical representation of the author's geolocation. The software requirements are described based on the analysis of past approaches using BERT, and the tasks of this work are set. The main functionality and special requirements for the implementation of algorithms, architecture, and auxiliary software tools are defined.

Section 2 is devoted to modelling the neural network architecture and designing multitasking machine learning algorithms for calculating loss functions of non-standard output format. The architecture of the Developer's software and the User's software in the form of the Telegram bot server side is described to demonstrate the work of the best trained model. The division of the modular architecture into classes in Python and the distribution of functionality to the corresponding utilities are considered. The proposed approach uses natural language processing (NLP) neural networks to estimate location in the form of coordinate pairs (longitude, latitude) and joint Gaussian Mixture Models (GMMs) with a restriction of

the output parameter responsible for the coefficient of the spherical covariance matrix.

In Section 3, we review past work in the subject area in terms of commonly used metrics of location prediction performance (accuracy). The performance metrics show that the average error is less than 30 km at the global level and less than 15 km at the US level for models trained and evaluated on the textual variables of tweet content (text) and their metadata context (user, location). Manual testing of training, evaluation and location prediction in the roles of Developer and User is described.

Finally, Section 4 is devoted only to the deployment of the server side of the Telegram bot, as the Developer's software is designed to run locally.

The software was implemented on a high-performance cluster based on a Unix-based OS without a graphical user interface.

The results of the work were tested at the level of the Director of the European Laboratory for Learning and Intelligent Systems (ELLIS) Christroph H. Lampert and submitted for publication in the Journal of Spatial Information Science (JOSIS).

KEYWORDS: GEOLOCATION PREDICTION, TRANSFORMERS, TWITTER DATASET, MACHINE LEARNING, REGRES- SION TASK, GAUSSIAN MIXTURE MODEL, MULTITASK LEARNING, TELEGRAM BOT, NUMERICAL DATA ANALYSIS, NATURAL LANGUAGE PROCESSING.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9116.045490.01.91	Технічне завдання	16	
3	А4	КПІ.ІТ-9116.045490.02.81	Пояснювальна записка	105	
4	А4	КПІ.ІТ-9116.045490.03.12	Текст програми	56	
5	А4	КПІ.ІТ-9116.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-9116.045490.05.34	Керівництво користувача	12	
7	А3	КПІ.ІТ-9116.045490.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІТ-9116.045490.06.99	Схема структурна діяльності	1	
9	А3	КПІ.ІТ-9116.045490.08.99	Схема структурна компонентів	1	
			програмного забезпечення		

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9116.045490.00.90		
Розроб.		Луцай К. А.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Баклан І.В.					
Н.контр.		Головченко М.М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-91	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Алгоритм та програмне забезпечення для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT

Технічне завдання

КПІ.ІТ-9116.045490.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ігор БАКЛАН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Катерина ЛУЦАЙ

Київ – 2023

Зміст

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Алгоритм та програмне забезпечення для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT.

Галузь застосування: аналіз текстових даних для отримання місцезнаходження

Наведене технічне завдання поширюється на розробку алгоритмів та ПЗ GeoBERT КПІ.ІТ-9116.045490, котра використовується для навчання та оцінки нейронних мереж для прогнозування місцезнаходження за текстом користувача та призначена для аналізу великих масивів коротких текстових даних, таких як записи у соціальних мережах, з метою визначення місцезнаходження автора.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки GeoBERT є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для аналізу великих масивів коротких текстових даних з метою визначення місцезнаходження авторів.

Метою розробки є підвищення точності прогнозів місцезнаходження за текстом, покращення алгоритмів навчання та оцінки нейронних мереж на основі BERT для обробки текстових даних, таких як записи користувачів соціальних мереж, покращення алгоритмів прогнозування місцезнаходження користувачів Twitter в рамках використання BERT-моделей на даних Twitter.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– Надання інформації про модель (посилання на вихідних код, репозиторій моделі, та статтю) як показано на Рисунку 4.1;

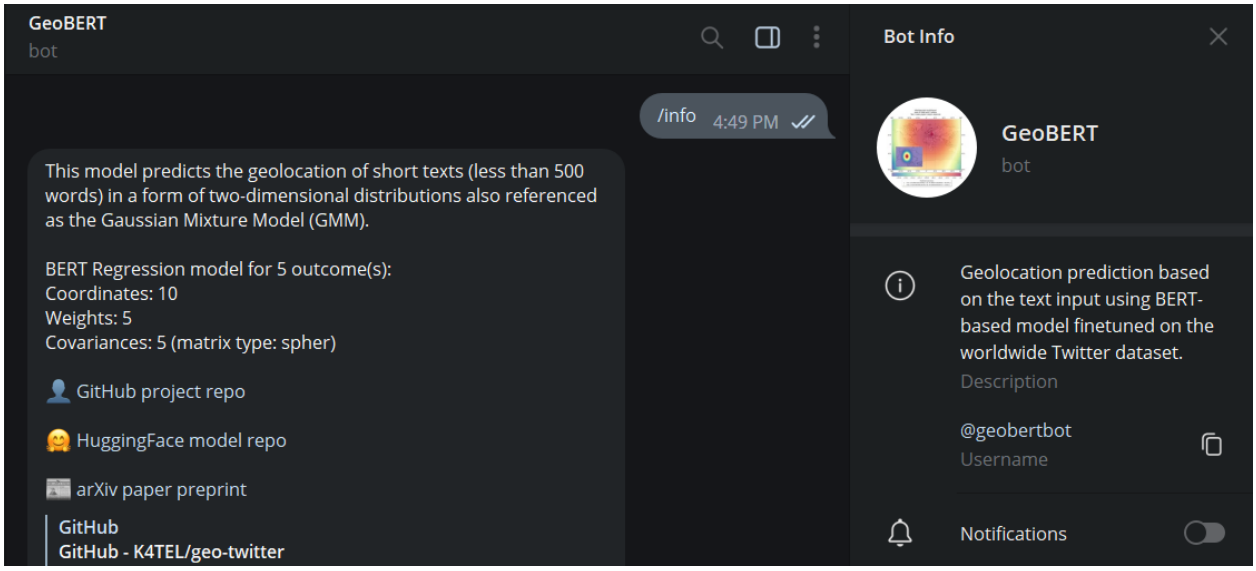


Рисунок 4.1 – Прототип екранної форми «info» з описом моделі

– Прогнозування місцезнаходження за текстом користувача у вигляді зображення двовимірного розподілу GMM на мапі та чисельно-текстового представлення результату як показано на Рисунку 4.2.

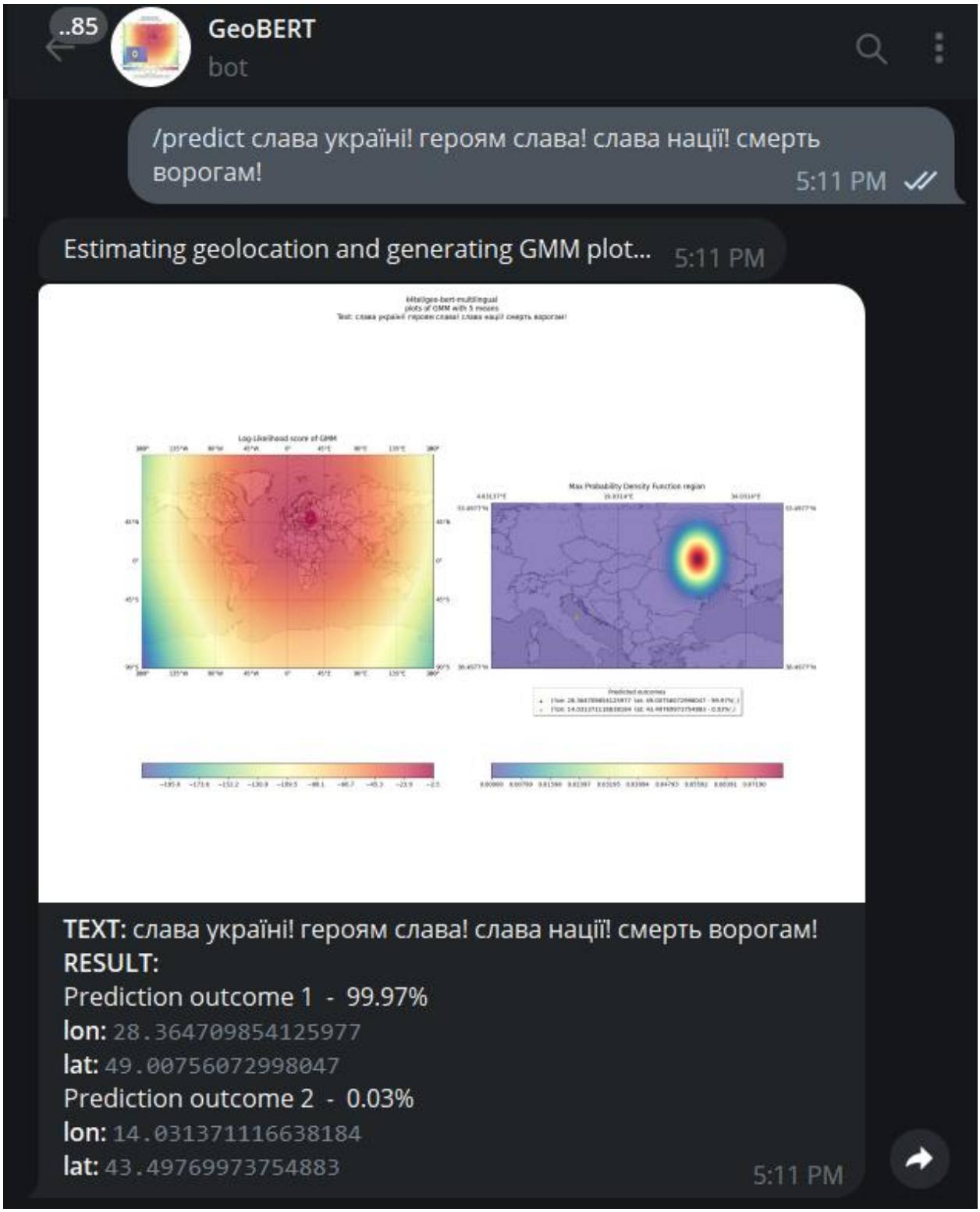


Рисунок 4.2 – Прототип екранної форми «predict» за текстовим аргументом

4.1.2 Для користувача:

– Стандартна команда «/start» для початку роботи з Telegram ботом;
– Команда «/info» для отримання інформації про модель;

– Команда «/predict <text>» для прогнозування місцезнаходження за введеним текстом.

4.1.3 Для адміністратора системи:

– Навчання власних моделей на датасеті вказаного формату;
– Оцінка власних моделей на датасеті вказаного формату;
– Візуалізація статистики помилок результатів прогнозу;
– Перегляд метрик точності навчання моделей;
– Прогнозування місцезнаходження за текстом використовуючи власну навчену модель;
– Формування вхідних змінних завантажувача даних моделі з датасету вказаного формату;
– Гнучка параметризація модифікацій архітектури моделей.

4.1.4 Додаткові вимоги:

– Можливість масштабування системи залежно від розміру датасету;
– Можливість налаштування системи під датасети різного рівня географічної деталізації (всесвітній, країна, місто і т.д.) з різницею у мовній специфікації;
– Можливість збереження проміжних та фінальних файлів моделей при навчанні;
– Можливість запуску ПЗ без графічного інтерфейсу та контроль через консоль для Розробника;
– Можливість збереження запитів та відповідей у БД Телеграм боту.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних логів Телеграм боту шляхом фільтрації вхідного тексту користувача. Забезпечити можливість відновлення навчання з контрольних точок (файлів моделей) у випадку збоїв.

Забезпечити сталий результат прогнозу місцезнаходження для однакового тексту користувача.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на серверах без графічного інтерфейсу користувача.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 4 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт;
– об’єм пам’яті: 10 Гб.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7;
– об‘єм ОЗП: 32 Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт;
– об’єм пам’яті: 1 Тб;
– об’єм відеопам’яті: 12 Гб;
– тип графічної карти: NVIDIA GeForce GTX 1080 Ti.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Unix.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі:

– файл датасету табличного формату з колонками “lon”, “lat” (чисельного типу), “text”, “user” та “place” (текстового типу) збережений у форматі .jsonl;

– файл контрольної точки навчання моделі у форматі .cpth;

– файл датасету табличного формату з колонками “lon”, “lat” (чисельного типу), сформованих вхідних змінних моделі (текстового типу), “mean”, “weight” та “cov” (чисельного типу) для усіх M піків GMM результату прогнозу місцезнаходження збережений у форматі .jsonl

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі:

– файл датасету табличного формату з колонками “lon”, “lat” (чисельного типу), сформованих вхідних змінних моделі (текстового типу), “mean”, “weight” та “cov” (чисельного типу) для усіх M піків GMM результату прогнозу місцезнаходження збережений у форматі .jsonl;

– файли статистики результатів оцінки моделі на датасеті у вигляді зображень графіків у форматі .png;

– файл статистики результатів оцінки моделі на датасеті записана у чисельно-текстовому вигляді у форматі .txt;

– файл зображення прогнозу місцезнаходження на мапі для єдиного тексту у форматі .png;

– файл контрольної точки навчання моделі у форматі .cpth;

– файл моделі у форматі .pth.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Python з огляду на наявність готових бібліотек для вирішення поставлених задач. Крім цього використати bash скрипти для консольних інтерфейсів на базі ОС Linux з огляду на умову використання ПЗ Розробника на сервері без графічного інтерфейсу користувача.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі PyCharm та використати віртуальне середовище Python у яке потрібно встановити допоміжні засоби розробки, такі як бібліотеки для машинного навчання, аналізу даних, роботи з датасетами, тощо.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді єдиного репозиторію GitHub де розміщено ПЗ Розробника з інструкціями до запуску навчання та оцінки власних моделей. Для використання вже навчених моделей з метою прогнозування місцезнаходження за текстом має бути надано мінімальний набір функцій та класів в окремій гільці проекту. Крім цього, серверна частина інтерфейсу Користувача у вигляді Телеграм боту має бути надана у вигляді набору файлів для розгортання Docker image та запуску Python скриптів.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Використати ймовірнісний тип прогнозу місцезнаходження у вигляді сумісних двовимірних розподілів на мапі в додаток до стандартної форми прогнозу у вигляді пари координат (довгота, широта). Для таких розподілів розробити алгоритми машинного навчання з моделюванням функцій втрат та архітектури нейронної мережі.

Використати багатомовну базову модель BERT з метою обробки датасетів всесвітнього масштабу.

Використати загальноприйняті метрики точності прогнозування місцезнаходження для порівняння запропонованої моделі з попередніми роботами.

Використати усі вхідні дані твітів у навчальних датасетах Твітера пов’язані з місцезнаходженням для багатофакторного аналізу та збільшення точності прогнозування.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача;
– керівництво програміста (розробника);

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна станів програмного забезпечення;
– архітектура програмного забезпечення;

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.03	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.03	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.04	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.04	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.04	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.04	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.04	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Алгоритм та програмне забезпечення для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT

КПІ.ІТ-9116.045490.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	– Integrated Development Environment – інтегроване середовище розробки.
API	– Application programming interface, прикладний програмний Інтерфейс
UI	– User interface
JSON	– JavaScript Object Notation – запис об'єктів JavaScript
ER	– Entity-Relation diagram
ОС	– Операційна система.
WGS84	– World Geodetic System 1984 – в геодезії тривимірна система координат для встановлення розташування на поверхні Землі.
GMM	– Gaussian mixture model – сумісна модель Гауса
PDF	– Probability density function – щільність ймовірності
LIW	– Location indicative word – слово-вказівник розташування
NLP	– Natural Language Processing – обробка природної мови
POI	– Point of interest – точка інтересу
GSOP	– Geospatial single outcome prediction – геопросторове прогнозування єдиного результату
GMOP	– Geospatial multiple outcome prediction – геопросторове прогнозування багатьох результатів
PSOP	– Probabilistic single outcome prediction – ймовірнісне прогнозування єдиного результату
PMOP	– Probabilistic single outcome prediction – ймовірнісне прогнозування багатьох результатів
KF	– Key Feature – основна змінна
MF	– Minor Feature – другорядна змінна
GPU	– Graphics processing unit – графічний процесор
CPU	– Central processing unit – центральний процесо
BERT	– Bidirectional Encoder Representations from Transformers – двоспрямовані кодувальні представлення з трансформерів
БД	– База даних.

SED – Squared Euclidean Distance – квадрат евклідової відстані

LBSP – Lower-Bounded SoftPlus – нижня межа SoftPlus

NLLH – Negative Log-LikeliHood – від’ємний логарифм правдоподібності

SM – SoftMax

WLC – Weighted Linear Combination – зважена лінійна комбінація

LSTM – Long short-term memory – довга короткочасна пам'ять

ВСТУП

Останніми роками багато досліджень було присвячено обробці великих масивів коротких текстових даних, таких як пости в соціальних мережах, з метою вилучення геолокації (місцезнаходження). З огляду на те що для обробки тексту та зображень все частіше використовуються нейронні мережі, а їх ефективність перевищує відомі алгоритми за швидкістю та точністю, було вирішено використати моделі BERT модифікованої архітектури для прогнозування місцезнаходження за текстовими даними. Даний тип моделей вже був використаний для прогнозування місцезнаходження у попередніх роботах, але запропонований підхід надає більш ефективний та точний спосіб вирішення поставленої задачі.

Навчання розроблених моделей відбувалося на датасеті Твітера що надає інформацію про місцезнаходження автора твіту в його метаданих. Це дозволило створити багатомовну модель навчену багатозадачними методами обробки вхідних змінних на твітах з усього світу для вирішення задачі регресії до параметрів розподілу GMM за контентом твіта та контекстом користувача.

Мета цієї роботи полягала в тому, щоб надати рішення, яке дозволить користувачам точно налаштовувати BERT-моделі запропонованого дизайну на власних наборах даних без необхідності використання зовнішніх баз знань, таких як географічні довідники. Незважаючи на те, що лише 26% користувачів надають інформацію про місцезнаходження у своїх профілях згідно з [4], а також на те, що згенеровані користувачем дані можуть бути зашумленими, запропоновані процедури багатозадачного машинного навчання підвищили геопросторову точність і призвели до того, що медіанна похибка склала менше 30 км для всесвітнього датасету.

Крім цього результатом роботи став користувацький інтерфейс у вигляді Телеграм боту для демонстрації прогнозування місцезнаходження за вхідним текстом що використовує кращу з навчених моделей.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Аналіз місцезнаходження надає дані для персоналізації, дозволяючи зрозуміти, як користувачі соціальних мереж ставляться до певної теми чи проблеми в конкретному регіоні. Таким чином, він допомагає соціальним наукам виявляти закономірності соціальної динаміки в певних сферах чи регіонах. Сюди входять питання, пов'язані з громадським здоров'ям, такі як вакцинація або наслідки пандемій [37], а також можливі уявлення про демографічні характеристики прихильників кандидата [1], які є важливими під час президентських виборів [41]. Крім того, аналіз місцезнаходження може бути корисним в урядових цілях у разі стихійних лих та управління кризовими ситуаціями, оскільки це може зменшити час реагування та допомогти краще розподілити ресурси. Крім того, у різних сферах бізнесу, включаючи роздрібну торгівлю, нерухомість і маркетинг [18], інформація про місцезнаходження дає змогу краще зрозуміти думку людей про певний бренд, продукт чи послугу в конкретному географічному регіоні.

Twitter, будучи широко використовуваною соціальною мережею в Інтернеті, накопичує великий обсяг різноманітних даних з високою швидкістю, включаючи короткі та невпорядковані твіти, широку мережу користувачів та багату контекстну інформацію як користувачів так і твітів. Ці дані слугують вхідними для вивчення трьох поширених проблем геолокації, таких як домашнє місцезнаходження користувача, місцезнаходження твіту та прогнозування згаданого місцезнаходження [44].

Розв'язання проблеми прогнозування місцезнаходження твіту вимагає правдивого місцезнаходження, яке зазвичай витягується з мета-даних твіту. У Твіттері лише 1-2% твітів мають геотеги з координатами довготи і широти [26], які також супроводжуються точним описом місця (країна, місто, тип та назва населеного пункту). Таким чином, існує два типи визначення місцезнаходження – пара числових координат і текстова мітка класу – які

передбачають, що тип завдання буде або числовою регресією, або класифікацією за текстовою міткою. Остання зазвичай застосовується для відносно невеликих задач, таких як прогнозування місцезнаходження на рівні міста [20]. Однак це дослідження фокусується на проблемах масштабу країни та глобального світового рівня, тому перший метод, що використовує координати довготи та широти як істинні, був обраний як більш доречний. Оскільки геолокаційні дані Twitter зберігаються у стандартному форматі системи координат, яка називається World Geodetic System 1984 року (WGS84), для відображення справжнього місцезнаходження твітів було використано ту ж саму геодезичну систему координат. Хоча WGS84 є безперервною глобальною системою координат, при відображенні даних на двовимірній карті, наприклад, у проекції Меркатора, числові значення координат залишаються в строго визначених діапазонах:

$$Y = (y_{lon}, y_{lat});\ y_{lon} \in [-180, 180];\ \ y_{lat} \in [-90, 90]$$

Рішення розв'язувати задачу числової регресії, а не класифікації міток за допомогою моделей на основі BERT, було також підтримано результатами експерименту, представленими в роботі Шеррера [33], які показали, що регресія перевершила класифікацію за показниками точності геостатування (середня та медіанна відстань помилки) для обмежених наборів даних Twitter боснійською, хорватською, чорногорською, сербською та нідерландською мовами.

Існує багато дослідницьких проектів, присвячених пошуку інформації у простому тексті для оцінки геолокації за допомогою нейронних мереж. З точки зору раніше згаданих проблем геолокації, які зазвичай асоціюються з даними Твіттера, цей розділ охоплює роботи, пов'язані з визначенням місцезнаходження твітів і прогнозуванням домашнього місцезнаходження користувача. Деякі рішення базуються виключно на тексті, а деякі застосовують гібридний підхід, використовуючи мережу друзів і згадок користувача на додаток до NLP методів. Хоча багато досліджень, присвячених прогнозуванню домашнього місцезнаходження користувачів, використовують

мережу зв'язків користувачів (наприклад, фоловерів, відповідей, згадок) для досягнення більшої точності [46] [45] [43], ця робота зосереджена в першу чергу на аналізі текстових даних. Тому для вирішення проблеми прогнозування місцезнаходження твітів використовується лише контент твіту (сам текст) та контекст твіту (метадані твіта та користувача). Крім того, згруповані за користувачами набори передбачень за твітами у вигляді двовимірних розподілів GMM були використані для вирішення задачі передбачення домашнього місцезнаходження користувача.

Загалом, існує три категорії деталізації місцезнаходження: адміністративні регіони, географічні сітки та географічні координати. Що стосується місцезнаходження твітів, то замість адміністративних регіонів або сіток, як правило, використовуються точки інтересу (POI) або координати, які відображають місцезнаходження твітів.

1.2 Змістовний опис і аналіз предметної області

Найпростіший підхід полягає в аналізі природних мов, присутніх у публікаціях у соціальних мережах. Існує багато проектів, які обробляють документи або резюме твітів профілю користувача для прогнозування місцезнаходження автора [38] [30] [39] [15] [23]. Ці роботи застосовують географічні сітки для поділу земної поверхні на регіони різного розміру таким чином, що густонаселені райони розбиваються на менші частини, тоді як більш розріджені в населенні райони стають великими комірками сітки. Ранні зусилля [12] [30] були в основному зосереджені на видобуванні орієнтовної інформації з контенту користувачів, покладаючись на слова-вказівники місцезнаходження (LIWs), які можуть пов'язати користувачів з їхніми домашніми адресами, на основі різних методів NLP (тематичних і статистичних моделей) [46]. Наприклад, Рахімі витягує ознаки наборів слів з дописів користувачів [28]; а Вінг і Болдрідж оцінюють розподіл слів для різних регіонів [38]. Інші словоцентричні роботи [31] [3] [21] [34] [25] також зосереджуються на фільтрації LIWs з тексту та використанні довідників, що

включають не лише назви міст/країн, але й діалектні терміни, для визначення географічних індексів. У той час як такі методики припускають наявність заздалегідь визначеного набору географічних назв та їх географічних координат, Рахімі в роботі [27] пропонує нейромережевий підхід до кодування таких слів і словосполучень у неперервному двовимірному просторі. У найстарішій з релевантних робіт Ейзенштейн представляє подібні некеровані (unsupervised) моделі, які базуються на темах [9] [10].

Інший спосіб прогнозування геолокації для постів у соціальних мережах ґрунтується на мережі користувачів, що висвітлено в багатьох попередніх роботах [40] [19] [17] [5] [22] [32]. Відповідно, деякі роботи пропонують гібридний підхід, що використовує як контент, так і мережу для вирішення задачі прогнозування домашнього місцезнаходження користувача [29] [8] [46] [45] [43]. Інші дослідники також використовують метадані на додаток до мережевих і текстових даних для своїх моделей [7] [24] [14]. Використання контекстних даних (таких як геотеги та інші метадані, пов'язані з твітами/користувачами) у попередніх роботах досліджено слабко через недостатню кількість даних, отриманих із загальнодоступних джерел наборів даних Twitter з геотегами.

З точки зору прогнозування геолокації окремого твіту, деякі автори запропонували ймовірнісні моделі, що вирішують завдання класифікації на рівні країни/міста [11] [16]. Крім того, Юань у роботі [42] також врахував часовий аспект і мобільність користувачів для оцінки джерела розташування кожного твіту.

Важливо, що методологія Придгорського близька до цієї роботи, оскільки вона передбачала використання сумісних моделей Гауса (GMM) для прогнозування геолокації замість простого прогнозування координат [26]. Інше дослідження, в якому використовувався гібридний підхід, також оцінювало геолокацію за допомогою GMM, а не координат [2]. У цьому дослідженні текст і мережеві ознаки використовувалися разом як предиктори

для остаточної оцінки. Метрики ефективності, використані в обох роботах, були прийняті для оцінки запропонованих імовірнісних моделей.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації алгоритмів та програмного забезпечення для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Більшість раніше згаданих робіт використовують різні підходи на основі нейронних мереж для NLP, але жодна з них не застосовувала BERT для оцінки геолокації. Відносно нова модель BERT (випущена в 2018 році [6]) призначена для задач класифікації, і деякі роботи вже використовували її для прогнозування назв країн/міст. Наприклад, Віллегас у роботі [36] прогнозує тип місця або точку інтересу (POI), використовуючи англомовну BERT. Крім того, Лі в роботі [20] зосередився на прогнозуванні в масштабі міста для наборів даних Twitter Мельбурна та Сінгапуру. Модель, представлена Лі, приймала на вхід текст і метадані та видавала результат у вигляді ймовірностей для набору класів POI.

У дослідженні, проведеному Сіманджунтаком [35], завдання прогнозування домашнього місцезнаходження користувачів Twitter вивчалося за допомогою моделей довготривалої та короткотривалої пам'яті (LSTM) і BERT. Набір даних, що використовувався в експерименті, складався з текстового контенту та контексту метаданих користувача твітів індонезійською мовою. Однак обмеження на вхідні дані в 512 токенів, накладені базовими BERT-моделями, створили проблему з точною класифікацією домашнього місцезнаходження користувача, оскільки об'єднання всіх твітів користувача в один текст призвело б до перевищення

розміру вхідних даних над встановленим обмеженням. В результаті, прогнозування домашнього місцезнаходження користувачів здійснювалося на основі кожного твіту, і було помічено, що підхід більшості голосів часто (42% випадків) призводив до неправильної класифікації домашнього місцезнаходження користувача.

Враховуючи, що базова модель має обмежену вхідну ємність у 512 токенів (слів), обробляти великі текстові масиви, які включають усі твіти користувачів, неможливо. Хоча, як повідомляється в [35], лише завдання прогнозування геолокації твітів може бути ефективно вирішене за допомогою базових моделей BERT, Simanjuntak оцінював домашнє місцезнаходження користувача за допомогою методу більшості голосів для набору точок, прогнозованих для твітів користувача. На відміну від нього, ця робота зосереджена на обробці менших (менше 300 слів) текстових вибірок та агрегуванні набору ймовірнісних прогнозів у вигляді GMM для оцінки набору найбільш ймовірних точок домашнього розташування користувача. Для цього було ітеративно розраховано поверхню функції щільності ймовірності (PDF) для всіх прогнозованих GMM на сітці, сформованій з усіх точок прогнозування для кожного твіту (піків GMM), потім розраховано середнє значення всіх значень PDF для кожної точки сітки, щоб отримати загальну оцінку для кожного користувача, і, нарешті, отримано локальні максимуми оцінюваної сітки. Такий підхід дозволив значно зменшити середню просторову похибку для задачі передбачення домашнього місцезнаходження користувача порівняно із задачею передбачення місцезнаходження твіту.

В іншій роботі Шеррер використовує BERT-моделі як для регресії, так і для класифікації на обмежених текстових БД боснійсько-хорватсько-чорногорсько-сербської та нідерландської мов [33]. Оцінка їхньої регресійної моделі на наборі даних, пов'язаному з німецькомовною Швейцарією, показала, що медіана та середня помилка відстані становить 21,20 та 30,60 км. Автори стверджують, що використання меншого словника токенізації та перетворення задачі геолокації на задачу класифікації загалом дало гірші результати. Вони

також роблять висновок, що налаштування гіперпараметрів не дало жодних послідовних покращень, а простий вибір оптимального номера епохи на основі даних про розвиток навчання виявився найкращим підходом до проблеми прогнозування місцезнаходження.

Результати представлені в [33], вказують на те, що регресія перевершує класифікацію в прогнозуванні геолокації за допомогою модифікованих BERT-моделей. Тому метою цієї роботи є покращення точності прогнозування місцезнаходження окремих твітів у вигляді множини можливих точок розташування, представлених географічними координатами (довгота, широта, вага) або двовимірними GMM (довгота, широта, вага, коваріація). Що стосується задачі прогнозування місцезнаходження користувача, то вихідні дані імовірнісних моделей з параметром коваріації зводяться до набору зважених пар координат (довгота, широта, вага).

У цій роботі рівень географічної деталізації було визначено як всесвітній, що охоплює весь спектр країн і мов, присутніх у стрічці Твіттера. Однак результати дослідження Шеррера [33] продемонстрували, що мовно-специфічні BERT-моделі явно перевершують свої багатомовні аналоги на більшості використаних наборів даних. У світлі цих висновків, модель на рівні країни для Сполучених Штатів була доопрацьована на американському наборі даних з використанням англомовної базової моделі BERT, яка технічно називається "bert-base-cased". Проте, глобальний набір даних вимагав використання багатомовної базової моделі BERT ("bert-base-multilingual-cased"), що, як і очікувалося, призвело до більш високих показників відстані похибки для прогнозування місцезнаходження для кожного твіту і користувача, як і очікувалося.

Важливою проблемою, яку ми вирішили, є представлення декількох мов, оскільки здатність обробляти кілька мов є ключовим фактором для поділу текстових даних на геопросторові регіони. Для вирішення цієї проблеми було використано багатомовну модель BERT, попередньо навчену на 104

найбільших мовах Вікіпедії, як відправну точку для навчання на глобальному наборі даних з геолокаційними мітками.

Базова BERT-модель призначена для навчання вирішення користувацького завдання, яким у цьому випадку була регресія до географічних координат (і параметрів GMM, таких як ваги та коваріація), що виводяться як тип завдання класифікації речень. Загалом, моделі BERT були попередньо навчені за допомогою маскованого мовного моделювання (MLM), яке дозволяє моделі вивчати двонаправлені представлення речень, і прогнозування наступного речення (NSP), щоб вловити зв'язки між реченнями. Оскільки шари BERT є ієрархічними, ранні шари BERT вивчають більш загальні лінгвістичні закономірності, такі як відмінності між багатомовними мовами та їхні загальні структури речень (у випадку багатомовної моделі BERT). Тоді як пізніші шари вивчають більш специфічні для конкретних завдань закономірності, в даному випадку – асоціації між точками на мапі світу і конкретними термінами, які називаються LIW, а також текстові конструкції і лінгвістичні закономірності, які зазвичай використовуються в певних географічних регіонах.

Завданням цієї роботи було знайти заміну словникам, які використовувалися в попередніх словоцентричних дослідженнях. Довідник зазвичай містить записи про міста, містечка, села, пам'ятки, природні об'єкти та інші географічні об'єкти, а також інформацію про їхнє розташування, координати, висоту над рівнем моря тощо. Аналогічно, метадані Twitter, що зберігаються в полі "місця" твітів з геотегами, надають автоматично згенеровану інформацію про країну, код країни, тип місцевості, назву місцевості та повну назву місцевості. Оскільки типові вхідні дані моделі не повинні включати контекст "місця", який відноситься виключно до твітів з географічними тегами, такі метадані не повинні бути присутніми і в навчальних послідовностях. Тому було впроваджено багатозадачне навчання, щоб зберегти формат вхідних даних та одночасно забезпечити модель точними

географічними термінами, отриманими з метаданих "місця", пов'язаних з реальним місцезнаходженням.

Підхід до багатозадачного навчання передбачає окремі шари обгортки для кожної з текстових вхідних змінних і спільну для всіх ознак втрату на пакет під час процесу навчання моделі. Ключовим вхідним елементом для найкращої із запропонованих моделей був текстовий вміст твіту у поєднанні з контекстом, отриманим з профілю автора (ім'я користувача, опис та місцезнаходження), тоді як другорядний вхідний елемент був присвячений виключно контекстним даним, отриманим з поля "місця". Враховуючи, що оцінювання проводилося на типових вхідних даних моделі ("текст" і "користувач"), моделі, навчені з описаним налаштуванням ключової та другорядної змінних, показали менші просторові помилки порівняно з моделями, навченими без другорядної змінної, або моделями, навченими на комбінації всіх даних в одній текстовій ознаці ("текст", "користувач" і "місце").

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Основна мова програмування для роботи з більшістю нейронних мереж – Python – була обрана з огляду на зручність у використанні та різноманіття готових бібліотек.

Для паралельного тренування моделей на сервері з GPU було використано Slurm Workload Manager. Slurm – це безкоштовний планувальник завдань із відкритим вихідним кодом для Linux і Unix-подібних ядер, який використовується багатьма світовими суперкомп’ютерами та комп’ютерними кластерами. Був обраний з огляду на використання серверу з цим планувальником задач для навчання та оцінки моделей на GPU.

Для написання коду на Python та запуску тренування моделей на локальній машині з CPU було використано IDE PyCharm що забезпечує аналіз коду, графічний налагоджувач, інтегрований блок-тестер, та інтеграцію з

системами контролю версій. Була обрана Community версія як безкоштовна IDE, що надає достатній для роботи функціонал.

Для контролю версій та збереження коду у хмарному сховищі було використано git – розподілену система керування версіями файлів та спільної роботи. Для збереження коду ПЗ онлайн було використано GitHub та GitLab – вебсервіси для спільної розробки ПЗ. Вихідний код був наданий у відкритий доступ на особистому репозиторії GitHub, а під час розробки було використано закрите хмарне сховище інституту на GitLab.

Для тренування моделей було обрано PyTorch – фреймворк машинного навчання з відкритим вихідним кодом, заснований на мові програмування Python та бібліотеці Torch. Використовується для розробки та навчання моделей глибокого навчання на основі нейронних мереж. PyTorch має перевагу над іншими фреймворками глибокого навчання, такими як TensorFlow та Keras, оскільки ключові особливості PyTorch включають:

– повний функціонал для побудови моделей глибинного навчання, що зазвичай використовується розпізнавання зображень та NLP, що надає можливість модифікацій архітектури обгорткового шару лінійної регресії нейронної мережі;

– оптимізація для додатків, що використовують графічні (GPU) та центральні (CPU) процесори, що дозволяє швидше проводити навчання на багатомільйонних наборах даних в рамках кластеру з відеокартками Nvidia та тестувати алгоритми навчання на звичайних ПК використовуючи один скрипт;

– використання динамічної графіки обчислень, що дозволяє розробникам, науковцям та відладчикам нейронних мереж запускати та тестувати частину коду в режимі реального часу;

– тензори PyTorch схожі на масиви NumPy, але також можуть працювати на графічному процесорі NVIDIA з підтримкою CUDA, отже використовувати відеопа’мять GPU для розміщення та оперування моделями та їх вхідними/вихідними даними.

Для розміщення власних та завантаження вже готових моделей було використано платформу HuggingFace від спільноти штучного інтелекту яка надає інструменти для побудови, навчання та розгортання моделей машинного навчання на основі відкритого коду та технологій. Даний сервіс був обраний з огляду на зручність розміщення великих за розміром моделей, таких моделі на основі BERT.

Для завантаження базових моделей BERT з репозиторіїв HuggingFace було використано бібліотеку transformers що надає API та інструменти для легкого завантаження та навчання найсучасніших NLP моделей. Для легкого доступу та роботи з готовою моделлю було використано підмодуль Pipeline що дозволяє створювати користувацькі алгоритми попередньої та пост обробки входу та виходу даних моделі.

Для відстеження прогресу навчання моделей було використано TensorBoard – інструмент для забезпечення вимірювань та візуалізації, необхідних під час процесу машинного навчання. Він забезпечує інструменти, необхідні для експериментів з машинного навчання, включаючи відстеження та візуалізацію таких показників, як втрати та точність, візуалізацію графів змінних гіперпараметрів тестових запусків навчання моделей, та багато іншого. Сервіс був обраний з огляду на консольний інтерфейс для зчитування файлів логів з кластеру, та зручність візуалізації у вигляді веб форм доступних до локального та дистанційного розміщення, де присутні інструменти для аналізу експериментів.

Для обробки числових даних було використано модулі з відкритим вихідним кодом NumPy та SciPy що надають загальні математичні та числові процедури у вигляді попередньо скомпільованих швидких функцій. Дані модулі були вибрані для аналізу вихідних параметрів GMM з огляду на жадібність алгоритмів та швидкість обчислень готових математичних функцій що надають дані Python бібліотеки.

Для вирахування помилкової відстані у кілометрах було використано бібліотеку Geopy що спрощує обчислення географічних відстаней між двома

точками. Бібліотека була обрана для обчислення статистики оцінки моделей на датасеті результатів прогнозів моделі для вирахунку точної відстані втрат у кілометрах з метою використання у просторових метриках.

Для роботи з датасетами у вигляді таблиць було використано бібліотеку Pandas що надає різні структури даних і функції для легкої та інтуїтивно зрозумілої роботи з реляційними або позначеними даними. Біблотека була використана для завантаження датасетів Twitter та збереження результатів прогнозу моделей для оброблених моделлю датасетів. Для розбиття даних на тренувальні та тестові було використано стандартну функцію бібліотеки Scikit-learn.

Для відстеження навантаження на GPU та CPU було використано модулі GPutil та psutil що надають засоби для аналізу використання ресурсів машини. Використання даних утиліт було обумовлене оптимізацією розміру пакету вхідних даних та розміру датасету, що може буту оброблений за один запуск навчання моделі, під апаратні характеристики.

Для відображення результатів у графічному форматі було використано бібліотеки Matplotlib та Seaborn які дозволяють користувачам створювати статичні, анімовані та інтерактивні візуалізації на Python. Зокрема були використані для візуалізації статистики просторових похибок та окремих результатів прогнозу за текстом. В додаток до цього було використано Basemap модуль що дозволяє генерувати світову мапу різної деталізації, що стало в нагоді при зображенні двовимірних розподілів GMM та наборів точок прогнозу моделі.

Для реалізації користувацького інтерфейсу у консолі було використано модуль argparse який полегшує написання зручних інтерфейсів командного рядка. Застосовуючи argparse, користувач програми може надавати значення змінним під час виконання без необхідності входити в код і вносити зміни в скрипт що полегшує параметризацію моделей. Даний модель було застосовано з огляду на розповсюдженість у Python скриптах для роботи на машинах без графічного інтерфейсу, таких як сервер (кластер).

Для реалізації користувацького інтерфейсу у вигляді Телеграм боту було використано бібліотеку requests що дозволяє викликати Telegram Bot API з метою отримання та відправки повідомлень у чат користувача з ботом. Дана бібліотека дозволила відправляти файли зображень, та отримувати текстові повідомлення користувачів.

Для розгортання серверної частини Телеграм боту було використано Docker – відкриту платформу для розробки, доставки та запуску програм, що спрощує процес створення, запуску, керування та розповсюдження додатків шляхом віртуалізації операційної системи комп'ютера, на якому він встановлений і працює. Дана платформа була використана з огляду на зручність ізоляції image серверного додатку в рамках системи.

Для збереження історії повідомлень Телеграм боту було використано SQLite – програмну бібліотеку, яка забезпечує автономну, безсерверну, транзакційну СУБД SQL у вигляді єдиного файлу що містить усі таблиці та індекси БД. Було розроблено лише одну таблицю для збереження логів запитів та відповідей Телеграм боту.

1.3.3 Аналіз відомих програмних продуктів

Відомих програмних продуктів для прогнозування місцезнаходження за текстом у відкритому доступі не існує оскільки такі алгоритми найчастіше розглядаються в рамках наукових робіт, що пропонують власні методології для вирішення поставленої задачі, а не готові рішення ПЗ. Тож аналіз відомих програмних продуктів має сенс лише в рамках порівняння точності моделей на базі загальнодоступних датасетів та метрик. Метрики ефективності поділяються на геопросторові та імовірнісні, перші є універсальними для всіх моделей та будуть використані далі у цій роботі для порівняння з попередніми дослідженнями як описано у Розділі 3.

1.4 Аналіз вимог до програмного забезпечення

Головними функціями програмного забезпечення є навчання, оцінка та використання нейронної мережі на базі BERT для прогнозування місцезнаходженням методами NLP. Варіанти використання ПЗ наведені на у графічному матеріалі «Схема структурна варіантів використання».

В Таблицях 1.1 – 1.9 наведені варіанти використання програмного забезпечення.

Таблиця 1.1 – Варіант використання UC-1

Use case name	Навчання моделі
Use case ID	UC-01
Goals	Навчання моделі для вирішення завдання прогнозування місцезнаходження
Actors	Розробник
Trigger	Розробник хоче розпочати навчання моделі з консолі
Pre-conditions	ПЗ було правильно встановлено за інструкцією користувача, параметри моделі задані правильно, файл датасету зчитано правильно
Flow of Events	Розробник ініціює навчання моделі на локальній машині за допомогою запуску скрипта з консолі. Розробник спостерігає за виводом у консолі під час навчання моделі, де відображаються поточні метрики точності та оцінки наприкінці епох. Чекпоінти та фінальна версія кращої моделі записуються у локальні файли формату .pth
Extension	У випадку виникнення помилок ініціалізації моделі навчання не розпочинається. Якщо у наявності недостатньо оперативної пам’яті навчання буде зупинене.
Post-Condition	Натреновану модель збережено у локальний файл, логи метрик під час навчання збережено у окрему папку.

Таблиця 1.2 – Варіант використання UC-2

Use case name	Оцінка моделі
Use case ID	UC-02
Goals	Оцінка моделі для вирішення завдання прогнозування місцезнаходження
Actors	Розробник
Trigger	Розробник хоче розпочати оцінку моделі з консолі
Pre-conditions	ПЗ було правильно встановлено за інструкцією користувача, параметри моделі задані правильно, файл датасету зчитано правильно
Flow of Events	Розробник ініціює оцінку моделі на локальній машині за допомогою запуску скрипта з консолі. Розробник спостерігає за виводом у консолі під час оцінки моделі, де відображаються поточні метрики точності та статистика результатів.
Extension	У випадку виникнення помилок ініціалізації оцінка не розпочинається. Якщо у наявності недостатньо оперативної пам’яті оцінку буде зупинене.
Post-Condition	Оцінений датасет та результати прогнозування збережено у локальний файл формату .jsonl, статистику результатів збережено у вигляді зображень в окрему папку, та у текстовому вигляді в окремий .txt файл

Таблиця 1.3 – Варіант використання UC-3

Use case name	Прогноз локації
Use case ID	UC-03
Goals	Прогнозування місцезнаходження за текстом у форматі GMM
Actors	Розробник, Користувач
Trigger	Розробник хоче отримати прогноз локації з консолі, або Користувач хоче отримати прогноз локації з команди Телеграм боту
Pre-conditions	Модель з HuggingFace завантажено та правильно ініційовано, довжина тексту не менше 1 та не перевищує 512 токенів

Продовження Таблиці 1.3

Flow of Events	Розробник або Користувач вводять текст за допомогою консолі або як аргумент команди Телеграм боту, текст фільтрується від посилань та знаків пунктуації, токенізується, та подається у модель. Чисельний результат обробляється та зберігається у вигляді параметрів GMM
Extension	У випадку введення тексту неправильної довжини буде видано попередження
Post-Condition	Формується набір параметрів GMM (широта, довгота, коваріація, вага) асоційований з введеним текстом як форма результату прогнозування місцезнаходження

Таблиця 1.4 – Варіант використання UC-4

Use case name	Параметризація моделі
Use case ID	UC-04
Goals	Встановлення параметрів та гіперпараметрів моделі
Actors	Розробник
Trigger	Розробник хоче розпочати навчання або оцінку локальної моделі
Pre-conditions	Розробник має локальну копію проекту, а ПЗ було правильно встановлено за інструкцією користувача
Flow of Events	Розробник редагує параметри моделі прописані у файлі запуску та зберігає зміни перед запуском навчання/оцінки. Або налаштовує параметри за допомогою аргументів командного рядка при запуску з консолі
Extension	У випадку введення неправильних даних навчання/оцінка не розпочинається
Post-Condition	Ініціалізація обгортки моделі на базі заданих параметрів

Таблиця 1.5 – Варіант використання UC-5

Use case name	Моніторинг навчання моделі
Use case ID	UC-05
Goals	Відстеження прогресу навчання моделі за метриками втрат та точності у вигляді графіків
Actors	Розробник
Trigger	Розробник відкриває логи навчання моделей за допомогою бібліотеки TensorBoard
Pre-conditions	Папка логів навчання моделі не порожня, Розробник має акаунт доступу до TensorBoard
Flow of Events	Розробник встановив та увійшов у власний обліковий запис TensorBoard, Розробник завантажує папку з логами навчання моделі. Розробник відкриває посилання на візуальне представлення метрик навчання моделі
Extension	У випадку введення неправильного шляху до папки логів або відсутності логів у папці сторінка з графіками не буде завантажена
Post-Condition	Сторінка з графіками доступна до перегляду за посиланням

Таблиця 1.6 – Варіант використання UC-6

Use case name	Завантаження датасету для обробки моделлю
Use case ID	UC-06
Goals	Перетворення текстових даних табличного формату в завантажувач даних моделі у форматі числових тензорів
Actors	Розробник
Trigger	Розробник хоче завантажити певний файл датасету для подальшої обробки моделлю
Pre-conditions	Папка датасетів має вказаний файл. Розмітка датасету включає колонки “lon”, “lat”, “text”, “user”, “place”. Формат файлу датасету .jsonl

Продовження Таблиці 1.6

Flow of Events	Розробник вказав шлях до файлу датасету, та запускає процедуру навчання або оцінки моделі. Датасет розбивається на частини вказаного розміру, відбувається попередня обробка текстових даних (фільтрація), токенізація тексту в тензори, та формування завантажувача даних для моделі.
Extension	У випадку введення не коректного шляху до датасету буде видано попередження та завершено програму. У випадку неправильного читання датасету для перетворення у завантажувач тензорів програму буде зупинене
Post-Condition	Формується готовий до завантаження у модель набір пакетів тензорів

Таблиця 1.7 – Варіант використання UC-7

Use case name	Збереження результатів прогнозування локації
Use case ID	UC-07
Goals	Збереження результату прогнозування місцезнаходження у чисельному форматі GMM параметрів для елементів датасету у текстовому форматі
Actors	Розробник
Trigger	Завершено оцінку моделі на датасеті яку запустив Розробник
Pre-conditions	Розробник запустив оцінку моделі на датасеті правильного формату, параметри моделі були задані правильно, оцінка пройшла без помилок
Flow of Events	Розробник запустив процес оцінки моделі на датасеті, під час якого формується масив результатів для кожного твіта та відбувається перетворення вихідних змінних (чисельних даних) у параметри GMM. Завантажувач даних доходить до останньої партії тензорів, після чого сформований масив прогнозів місцезнаходження розбивається на колонки вихідного датасету “lon”, “lat”, “weight”, “cov” для кожного з M піків GMM. Сформована таблиця колонок доповнює оригінальний датасет та зберігається у форматі .jsonl

Продовження Таблиці 1.7

Extension	У випадку зупинення процесу оцінки моделі, або різної довжини оригінальних та спрогнозованих колонок датасет не буде збережено.
Post-Condition	У відповідну папку зберігається датасет зі згенерованими колонками результатів

Таблиця 1.8 – Варіант використання UC-8

Use case name	Збереження статистики результатів прогнозу моделі
Use case ID	UC-08
Goals	Отримання графічного та чисельного представлення статистики результатів
Actors	Розробник
Trigger	Розробник зазначає тип бажаного графіку статистики (розподіл щільності, або кумулятивнй розподіл) для попередньо сформованого датасету результатів.
Pre-conditions	Датасет результатів сформовано, тип графіку вказано до запуску оцінки або окремо після її завершення
Flow of Events	Розробник зазначає тип графіку для обчислення статистики датасету результатів. Відбувається перерахування дистанції у кілометрах для кожного твіта від його справжнього місцезнаходження до отриманих прогнозів враховуючи вагу кожного з M піків GMM. Формується зображення графіку на базі проведених розрахунків. Метрики датасету результатів записуються в текстовому форматі у окремий файл
Extension	У випадку невідповідності заданих параметрів моделі та датасету статистику не буде вирахувано правильно
Post-Condition	У відповідну папку зберігається зображення отриманого графіку. Метрики геопросторових та ймовірнісних похибок зберігаються у текстовому форматі в окремий файл.

Таблиця 1.9 – Варіант використання UC-9

Use case name	Перегляд результату прогнозу моделі на мапі
Use case ID	UC-09
Goals	Отримання графічного представлення прогнозу моделі
Actors	Користувач
Trigger	Користувач використовує модель для прогнозування місцезнаходження за текстом у Телеграм боті за допомогою команди «/predict»
Pre-conditions	Довжина тексту користувача не менше 1 і не перевищує 512 токенів (слів). Модель обробила текст та видала чисельний результат з якого формуються параметри GMM. Параметри моделі узгоджені з форматом отриманого результату.
Flow of Events	Користувач запускає Телеграм бот та відправляє текст для прогнозування місцезнаходження використовуючи команду «/predict» з текстом як аргументом. Модель обробляє текст та видає чисельний результат який обробляється для отримання параметрів GMM. Отриманий розподіл наноситься у вигляді кольорового спектру на мапі світу. Зображення відправляється з сервера до клієнта (чат Користувача з ботом).
Extension	У випадку неправильно введеного тексту зображення не буде отримано, буде відправлено попередження про помилку
Post-Condition	У відповідну папку зберігається зображення спектру отриманого розподілу вірогідності місцезнаходження. Користувач отримує зображення та текстовий підпис що включає чисельно-текстове представлення результату у чаті Телеграм боту.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій, в Таблицях 1.11 – 1.20 наведений опис функціональних вимог до ПЗ. Матрицю трасування вимог можна побачити на Рисунку 1.5.

Таблиця 1.10 – Модель функціональних вимог

№	Опис вимоги	Код	Пріоритет	Ризик
1	Параметризація навчання/оцінки моделі	FR-1	1	+
1.1	Встановлення набору ключових та другорядних змінних	REQ-1	1	-
1.2	Встановлення назви файлу датасету до завантаження	REQ-2	1	+
1.3	Встановлення розміру пакету завантажувача даних	REQ-3	1	+
1.4	Встановлення розміру випадкового вибірки даних	REQ-4	1	+
1.5	Встановлення ключа випадкового вибору записів	REQ-5	3	-
1.6	Встановлення назви файлу локально збереженої моделі	REQ-6	1	+
1.7	Встановлення типу базової моделі BERT	REQ-7	1	-
1.8	Встановлення використання параметрів ваги у прогнозі	REQ-8	1	-
1.9	Встановлення кількості точок результату прогнозу	REQ-9	1	-
1.10	Встановлення типу матриці коваріації GMM прогнозу	REQ-10	1	-
1.11	Встановлення типу оцінки (за твітом/користувачем)	REQ-11	2	-
1.12	Встановлення розміру набору оцінки в користувачах	REQ-12	2	+
1.13	Встановлення кількості епох навчання	REQ-13	1	-
1.14	Встановлення початкового/кінцевого кроку навчання	REQ-14	2	+

Продовження Таблиці 1.10

1.15	Встановлення типу планувальника навчання	REQ-15	2	+
1.16	Встановлення кроку логування метрик навчання	REQ-16	2	-
1.17	Встановлення пропорції тестового набору даних навчання	REQ-17	2	+
1.18	Встановлення типу обчислення функції тотальних втрат	REQ-18	2	-
1.19	Встановлення типу обчислення функції втрат вхідної змінної	REQ-19	1	-
1.20	Встановлення типу обчислення імовірнісної функції втрат	REQ-20	1	-
1.21	Встановлення типу обчислення геопросторової функції втрат	REQ-21	1	-
1.22	Встановлення збереження проміжних файлів моделі	REQ-22	2	+
2	Моніторинг метрик навчання моделі у реальному часі	FR-2	2	-
2.1	Перегляд метрик втрат навчального набору даних	REQ-23	2	-
2.1.1	Перегляд метрик імовірнісних втрат	REQ-24	2	-
2.1.2	Перегляд метрик геопросторових втрат	REQ-25	1	-
2.1.3	Перегляд метрик втрат другорядних змінних	REQ-26	2	-
2.1.4	Перегляд метрик втрат ключової змінної	REQ-27	2	-
2.2	Перегляд метрик втрат тестового набору даних	REQ-28	1	-
3	Завантаження набору даних Twitter з файлу у модель	FR-3	1	+
3.1	Зчитування файлу формату .jsonl	REQ-29	1	+
3.2.1	Обробка колонок текстових полів “text”, “user” та “place”	REQ-30	1	+

Продовження Таблиці 1.10

3.2.2	Обробка колонок числових координат “lon” та “lat” лейблів	REQ-31	1	+
3.3	Фільтрація записів користувачів-ботів (більше 20 в день)	REQ-32	2	-
3.4.1	Обрізання датасету до встановленої довжини	REQ-33	2	+
3.4.2	Вибірка записів для встановленого числа користувачів	REQ-34	2	+
3.5	Розподіл датасету на навчальний та тестовий набори даних	REQ-35	1	-
3.6	Формування вхідних змінних KF та MF з колонок датасету	REQ-36	1	-
3.7	Фільтрація тексту від пунктуації та посилань	REQ-37	2	-
3.8	Токенізація тексту стандартними методами BERT	REQ-38	1	-
3.9	Розподіл набору даних на пакети встановленої довжини	REQ-39	1	-
4	Збереження обробленого набору даних у файл	FR-4	1	+
4.1	Формування колонок текстових змінних KF та MF, та правдивої точки	REQ-40	2	-
4.2	Обробка початкових вихідних змінних моделі	REQ-41	1	+
4.2.1	Нормалізація вихідних параметрів ваги точок	REQ-42	1	-
4.2.2	Обмеження вихідного параметру матриці коваріації точки	REQ-43	1	+
4.3	Формування колонок числових параметрів прогнозу	REQ-44	1	+
5	Збереження статистики результатів у датасеті	FR-5	1	+
5.1	Обчислення метрик за місцезнаходженням користувача	REQ-45	2	+

Продовження Таблиці 1.10

5.2	Обчислення просторових метрик результатів оцінки датасету	REQ-46	1	-
5.2.1	Обчислення відстані похибки у кіломіетрах	REQ-47	1	-
5.2.2	Обчислення відсотку похибок нижче заданого порогу	REQ-48	2	-
5.3	Візуалізація розподілу густини логарифму відстані похибки	REQ-49	2	-
5.4	Візуалізація кумулятивного розподілу відстані похибки	REQ-50	2	-
5.5	Обчислення імовірнісних метрик результатів оцінки датасету	REQ-51	2	-
5.5.1	Обчислення кумулятивної відстані вибірки GMM у км	REQ-52	3	+
5.5.2	Обчислення площі покриття регіону 95% вірогідності GMM	REQ-53	3	-
5.5.3	Обчислення відсотку покриття регіоном 95% вірогідності	REQ-54	3	-
5.6	Збереження статистики метрик у чисельно-текстовому виді	REQ-55	1	-
6	Завантаження локально збереженої або розміщеної онлайн моделі	FR-6	1	+
6.1	Зчитування локального файлу моделі формату .pth	REQ-56	1	+
6.1.1	Зчитування збереженого словника станів моделі	REQ-57	1	+
6.2	Зчитування локального файлу моделі формату .cpth	REQ-58	2	+
6.2.1	Зчитування збереженого словника станів планувальника	REQ-59	2	+

Продовження Таблиці 1.10

6.2.2	Зчитування збереженого значення середніх втрат моделі	REQ-60	2	+
6.2.3	Зчитування збереженого номеру пройдених епох	REQ-61	2	+
6.3	Завантаження готової моделі з репозиторію HuggingFace	REQ-62	1	-
7	Візуалізація результату єдиного прогнозу	FR-7	1	+
7.1	Зображення точок прогнозу моделі за текстом	REQ-63	2	-
7.2	Зображення PDF та log-likelihood розподілів GMM прогнозу	REQ-64	1	-
7.3	Скорочення результату до точок зі значимою вагою	REQ-65	2	-
7.4	Підпис типу моделі та контенту вхідної текстової змінної	REQ-66	2	-
8	Гнучке підлаштування алгоритмів машинного навчання під модель	FR-8	1	+
8.1	Налаштування обчислення функції втрат згідно типу моделі	REQ-67	1	+
8.1.1	Налаштування типу функції (просторова/імовірнісна)	REQ-68	1	-
8.2	Налаштування втрат багатозадачних (KF та MF) моделей	REQ-69	1	-
8.2.1	Налаштування обчислення втрат ключової змінної KF	REQ-70	1	-
8.2.2	Налаштування обчислення втрат другорядних змінних MF	REQ-71	1	-
8.3	Налаштування специфіки обчислення просторових втрат	REQ-72	1	-

Продовження Таблиці 1.10

8.4	Налаштування специфіки обчислення імовірнісних втрат	REQ-73	1	-
8.4.1	Налаштування кількості вихідних параметрів коваріації	REQ-74	1	-
8.4.2	Налаштування обмеження параметру матриці коваріації	REQ-75	1	-
8.5	Налаштування обробки прогнозу зважених точок результату	REQ-76	1	-
8.6	Налаштування кроку логування метрик навчання	REQ-77	2	-
8.7	Налаштування обчислення тотальних втрат пакету	REQ-78	1	-
9	Гнучке підлаштування оцінки та обробки результатів під модель	FR-9	1	+
9.1	Налаштування кількості точок результату у прогнозі	REQ-79	1	-
9.1.1	Налаштування обробки набору зважених точок прогнозу	REQ-80	1	+
9.2	Налаштування типу прогнозу (просторовий/імовірнісний)	REQ-81	1	-
9.3	Налаштування обробки вихідних змінних прогнозу	REQ-82	1	+
10	Збереження результуючих файлів навчання/оцінки моделі	FR-10	1	+
10.1	Збереження фінальних та проміжних файлів навчання	REQ-83	1	+
10.1.1	Збереження логів втрат та точності навчання моделі	REQ-84	2	+

Продовження Таблиці 1.10

10.1.2	Збереження контрольних точок в кінці епохи навчання	REQ-85	2	+
10.1.3	Збереження фінального стану моделі після навчання	REQ-86	1	+
10.2	Збереження фінальних та проміжних файлів оцінки	REQ-87	1	+
10.2.1	Збереження датасету після оцінки моделлю	REQ-88	1	+
10.2.2	Збереження статистики результатів оцінки датасету	REQ-89	2	+
10.2.3	Збереження розподілів відстані похибки датасету	REQ-90	2	+

Таблиця 1.11 – Функціональна вимога FR-1

Назва	Параметризація навчання/оцінки моделі
Опис	Система повинна надавати можливість навчати/оцінювати власні моделі розробнику шляхом введення аргументів параметрів моделі у консолі або редагування скрипту запуску

Таблиця 1.12 – Функціональна вимога FR-2

Назва	Моніторинг метрик навчання моделі у реальному часі
Опис	Система повинна надавати можливість відстеження втрат та точності під час навчання моделі розробнику шляхом виведення статистики логів у консолі або через сторонній сервіс (TensorBoard)

Таблиця 1.13 – Функціональна вимога FR-3

Назва	Завантаження набору даних Twitter з файлу у модель
Опис	Система повинна надавати можливість завантажувати власні датасети розробнику шляхом вказання назви файлу у параметрах запуску навчання/оцінки

Таблиця 1.14 – Функціональна вимога FR-4

Назва	Збереження обробленого набору даних у файл
Опис	Система повинна надавати можливість зберігати оброблений датасет розробнику шляхом формування таблиці з числових результатів роботи моделі (параметрів GMM)

Таблиця 1.15 – Функціональна вимога FR-5

Назва	Збереження статистики результатів у датасеті
Опис	Система повинна надавати можливість отримати графічне представлення статистики похибок розробнику шляхом генерації зображень та підрахунку метрик по завершенню оцінки

Таблиця 1.16 – Функціональна вимога FR-6

Назва	Завантаження локально збереженої або розміщеної онлайн моделі
Опис	Система повинна надавати можливість завантажувати модель користувачеві шляхом вказання шляху до локального файлу або репозиторію hugging face з моделлю

Таблиця 1.17 – Функціональна вимога FR-7

Назва	Візуалізація результату єдиного прогнозу
Опис	Система повинна надавати можливість отримати візуальне представлення результату прогнозу користувачеві шляхом вказання тексту

Таблиця 1.18 – Функціональна вимога FR-8

Назва	Гнучке підлаштування алгоритмів машинного навчання під модель
Опис	Система повинна надавати можливість гнучкого налаштування датасету, функцій втрат та архітектури для навчання моделі згідно параметрів моделі вказаних розробником

Таблиця 1.19 – Функціональна вимога FR-9

Назва	Гнучке підлаштування оцінки та обробки результатів під модель
Опис	Система повинна надавати можливість гнучкого налаштування датасету, функцій обробки вихідних значень моделі та їх запису під час оцінки згідно параметрів моделі вказаних розробником

Таблиця 1.20 – Функціональна вимога FR-10

Назва	Збереження результуючих файлів навчання/оцінки моделі
Опис	Система повинна надавати можливість збереження зображень, файлів моделей, оброблених датасетів, чисельних метрик, та логів навчання розробнику шляхом автоматичного запису відповідних файлів під час навчання/оцінки моделі

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10
UC-1	X	X	X					X		X
UC-2	X		X	X	X	X			X	X
UC-3						X	X			
UC-4	X							X	X	
UC-5		X								
UC-6			X					X	X	
UC-7				X					X	X
UC-8					X				X	X
UC-9							X			

Рисунок 1.1 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Далі наведено нефункціональні вимоги ПЗ:

Масштабованість – при збільшенні розміру датасету продуктивність роботи моделі не зміниться, але час її навчання/оцінки збільшиться. Для серверної частини Телеграм боту збільшення робочого навантаження призведе до збільшення часу відгуку, але послідовна обробка великої множини заявок клієнтів не повинна призвести до зупинки серверної частини ПЗ.

Безпека та конфіденційність – вхідні дані користувачів Телеграм боту мають проходити фільтрацію перед збереженням у БД, а датасети використані для тренування та оцінки не мають бути опубліковані у загальний доступ з огляду на наявність персональної інформації користувачів Twitter у метаданих твітів.

Локалізація – для обробки датасетів всесвітнього глобального рівня потрібно навчати багатомовну базову модель BERT, а для підвищення точності прогнозів рекомендовано використовувати специфічні за мовою моделі в залежності від досліджуваної місцевості. Загалом ПЗ має англомовний інтерфейс без передбаченої локалізації.

Інтерфейси ПЗ – для Розробника достатньо консольного інтерфейсу без графічної оболонки для запуску навчання та оцінки моделей на сервері, для Користувача потрібно надати зручний та зрозумілий у використанні UI, наприклад Телеграм бот, для використання готової моделі з метою прогнозування місцезнаходження за вхідним текстом.

Апаратні та програмні вимоги – для роботи ПЗ достатньо апаратної платформи з наявними CPU та GPU (мінімальна вимога: 12-16 Гб пам’яті, наприклад, NVIDIA GeForce GTX 1080 Ti з драйвером CUDA версії 11.6 та вище) вузлами, 500 Гб комп’ютерної пам’яті для зберігання датасетів та файлів локальних моделей, ОС на базі Linux що підтримує Python версії 3.8 або вище. Для розгортання серверної частини Телеграм боту потрібно щоб програмна платформа була підключена до мережі інтернет та включала встановлений Docker та tmux сервіси.

Надійність – результати прогнозу моделі мають бути стабільними для однакових вхідних даних, поставлені експерименти з машинного навчання

мають бути відтворюваними без значної втрати точності результатів. Сервер Телеграм боту має витримувати малі та середні навантаження трафіку заявок від клієнтів протягом достатньо довгого часу як користувацький інтерфейс для демонстрації можливостей моделі.

Продуктивність – вимірюється точністю прогнозів місцезнаходження моделлю за мірою похибки у кілометрах або у градусах, має бути порівнюваною з попередніми роботами для визначення продуктивності запропонованих алгоритмів машинного навчання. Час обробки одного твіту моделлю та формування чисельного результату у формі параметрів GMM має бути ду;t коротким (менше 0.1 секунди), а час генерація зображення розподілу імовірності місцезнаходження на мапі не має перевищувати 30 секунд.

Юзабіліті – ПЗ має бути простим у налаштуванні під задачу та датасет Розробника, а функціонал Телеграм боту має бути інтуїтивно зрозумілим для Користувача.

Збереження даних – для збереження повідомлень користувачів та відгуку Телеграм боту слід впровадити мінімальну БД з логами кожного запиту та відповіді на нього.

Керування помилками – для консольного інтерфейсу Розробника слід впровадити обробку типових помилок параметризації, завантаження датасетів, завантаження та збереження моделей, генерації статистики результатів, тощо у вигляді виводу помилки та зупинення виконання програми. Для Телеграм боту слід впровадити обробку помилок без зупинки серверної частини застосунку та з виводом помилки у консоль.

Узгоджені стандарти – формат місцезнаходження задається стандартом WGS84 (довгота, широта) та мінімальними параметрами GMM (вага, коефіцієнт сферичної коваріації), формат датасету задається набором колонок (текст, користувач, місце) для подальшого формування текстових змінних згідно з типом моделі.

1.5 Постановка задачі

Основною метою цієї роботи було покращення точності прогнозування місцезнаходження у вигляді географічних точок та двовимірних розподілів на основі коротких текстів, оброблених за допомогою модифікованих BERT-моделей. З точки зору вхідних даних, які використовувалися для налаштування моделей, в цьому дослідженні використовувалася лише текстова інформація, така як вміст твіту (необроблений текст користувача) та контекст твіту (метадані, такі як інформація з профілю користувача та текстові описи геотегів). Модифікації архітектури для найкращих із запропонованих моделей стосувалися розміру вихідних векторів шарів обгортки, що реалізують лінійну регресію, для виведення параметрів, що визначають GMM (тобто координат, ваг та параметрів коваріації). Звідси випливають спеціальні процедури обчислення функції втрат і додаткова реалізація методів багатозадачного навчання для обробки декількох текстових змінних.

В рамках ПЗ очікується виконання описаних вище функціональних задач та реалізація повної моделі функціональних вимог, зокрема: зчитування та збереження відповідних файлів, налаштування алгоритмів під параметри Розробника, специфіка навчання, оцінки, та прогнозування місцезнаходження за текстом.

Висновки до розділу

Було описано предметну область та попередні роботи з прогнозування місцезнаходження користувача за текстовими даними. Проаналізовано схожі роботи що використовують моделі BERT та визначено найкращу стратегію для розробки власного ПЗ та модифікації архітектури нейронної мережі.

Було обрано завдання регресії до числових координат та двовимірних розподілів замість задачі класифікації звичної для такого типу моделей. Було вирішено зробити наголос на прогнозуванні місцезнаходження твіта, а не домашнього місцезнаходження користувача. Незважаючи на більшу

ефективність BERT моделей для окремих мов, було вирішено використовувати багатомовну попередньо навчену модель для обробки твітів з усього світу.

Було визначено три основні випадки використання майбутнього ПЗ: навчання та оцінка моделей на власних датасетах, використання готової моделі для прогнозування місцезнаходження окремих текстів. Також було визначено базові функціональні та нефункціональні вимоги ПЗ.

Було описано поставлену задачу покращення точності прогнозування місцезнаходження у форматі параметрів GMM з довільною кількістю піків для датасетів всесвітнього рівня з використанням модифікованої моделі на базі багатомовної версії BERT навченої на вирішення задачі регресії до безперервних числових значень (довгота, широта, коваріація, вага) за допомогою багатозадачних алгоритмів машинного навчання.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Загальним підходом до розв'язання задачі регресії для BERT-моделей є додавання щільного лінійного шару поверх вихідних токенів класифікації. Щоб врахувати багатозадачність навчання, для ключових і другорядних текстових змінних використовувалися окремі обгорткові шари. Враховуючи це, а також двовимірні розподіли GMM, що використовуються як вихідний формат імовірнісних моделей, у цій роботі запропоновано спеціальні процедури обчислення функції втрат для чотирьох типів моделей, визначених у Таблиці 2.3, на основі вихідного формату моделі. Процедуру навчання найкращої із запропонованих моделей продемонстровано у графічному матеріалі «Схема структурна компонентів програмного забезпечення» .

Регресійний шар використовувався для перетворення вектора кінцевих прихованих станів, що складається з 768 чисел, до заданої кількості числових виходів. Відповідно до завдання прогнозування місцезнаходження, необхідна кількість виходів була визначена як 2, що відповідають географічній точці (довгота, широта) в системі координат WGS84. Однак найкраща з запропонованих моделей мала 20 числових значень, які використовувалися як параметри для визначення прогнозованої GMM з 5 піків. Зауважимо, що моделі, які прогнозують одну пару координат, можна навчати, використовуючи стандартну функцію втрат середньоквадратичної помилки (MSE). Однак для більш складних вихідних форм, таких як GMM, потрібна спеціальна функція втрат, описана нижче.

Імовірнісні моделі прогнозування одного/кількох результатів (PSOP/PMOP) дещо перевершили прямий підхід геопросторових моделей прогнозування одного/кількох результатів (GSOP/GMOP) за показником відстані між медіанними помилками.

Етап налаштування гіперпараметрів охоплює такі параметри, як тип планувальника, мінімальний та максимальний крок навчання для обраного планувальника та кількість епох. Для поточної задачі було обрано косинусоїдальний тип з набору лінійних, циклічних, ступінчастих та плато-планувальників. Експерименти показали, що оптимальний діапазон зменшення швидкості навчання починається з 1e-5 і закінчується на 1e-6 в кінці останньої епохи навчання. Вхідну кількість епох було зменшено з 5 до 3 для набору даних, що складається з 2,7 млн навчальних вибірок, які в завантажувачі даних моделі групуються в пакети від 10 до 16 твітів. Таке рішення було прийнято через те, що після третьої епохи не спостерігалося значного зменшення відстані похибки як для тренувальних, так і для тестових даних. За замовчуванням, тестовий набір з 300,000 зразків оцінювався без поширення градієнта в кінці кожної епохи. Для навчання було застосовано описані вище параметри, що були постійними для всіх запропонованих моделей, а варіації стосувалися лише типу функції втрат, кількості результатів (точок прогнозування або піків GMM), типу коваріації та комбінації текстових ознак.

Набори даних, використані для навчання моделей, складалися з твітів, зібраних у 2020-2022 роках, які мали визначені об'єкти "координат" та "місця" в JSON твіту. Перед завантаженням у модель набори даних пройшли кілька етапів попередньої обробки, включаючи фільтрацію текстових рядків, перегрупування стовпців для формування текстових змінних, токенізацію та розбиття на тензорні завантажувачі даних заданого розміру пакетів. Модель оперувала виключно вхідними ідентифікаторами та масками уваги, що представляли текстові ознаки набору даних та їхніми числовими мітками, які відповідали координатам геолокації.

Загалом лише 5% усіх твітів в архівній базі даних Twitter, зібраних з 2020 по 2022 рік, мають координати геолокації, тобто кількість доступних зразків з геотегами становила приблизно 150 мільйонів. Було виявлено, що до 20% твітів з геотегами опубліковані користувачами-ботами, тобто

автоматизованими акаунтами, наприклад, газетами, прогнозами погоди, трекерами для відстеження літаків, тощо. Хоча набори даних для навчання містили твіти як реальних користувачів, так і ботів, набори даних для оцінювання були відфільтровані на основі припущення, що реальні користувачі не публікують більше 20 повідомлень на день. На етапі попередньої обробки вихідні дані були відфільтровані – об'єкти твіту були сконденсовані в текстовий контент, відповідний контекст метаданих і координати геолокації, що відображають справжнє місцезнаходження.

Таблиця 2.1 – Вміст текстових змінних, сформованого з набору даних розбитих полів JSON твітів

Назва змінної					**Колонки датасету**	**JSON поля**
ALL	TEXT-ONLY	USER-ONLY	GEO-ONLY	NON-GEO		
+	+			+	text	text
+			+	+	user	location, description, name, screen_name
+			+		place	country, place_type, location, name, full_name

Справжня геолокація для цього дослідження була визначена як пара координат "довгота-широта", пов'язана з кожним твітом з геотегом. Існує два внутрішніх JSON твіта, які описують інформацію про місцезнаходження твіту: "координати" та "місце". Перший присутній лише тоді, коли було вказано точне місцезнаходження, тому пара координат у форматі "довгота-широта" була зібрана шляхом парсингу лише об'єкта "координат" твіту. На відміну від цього, поле "місце" містить точний автоматично згенерований текстовий опис місцезнаходження, наприклад, назву країни чи міста, що було використано як другорядна змінна під час навчання моделей.

З метою навчання моделі на геопросторових ознаках, пов'язаних з твітом, окрім оригінального тексту твіту, були зібрані додаткові метадані. Такі метадані забезпечили контекст для змісту твіту і підвищили загальну точність моделей. Оскільки об'єкти кореневого рівня JSON твіту "place" і "user" мають кілька полів, що містять потенційно релевантні контекстні дані, ці текстові поля об'єднуються у відповідні стовпці рядкового типу набору даних, описаного в Таблиці 2.1. Загалом, кожен твіт завжди має пов'язану з ним інформацію "тексту" і "користувача", комбінація яких була використана як ключова навчальна змінна та вхідні дані для оцінки. Крім того, поле "місце" присутнє лише в твітах з географічними тегами, тому його було визначено як другорядну навчальну змінну, яка ігнорується під час оцінювання.

Таблиця 2.2 - Комбінації текстових змінних, що використовуються в текстовому (контент) та гібридному (контент і контекст) підходах

Тип даних	**Змінні навчання**		**Точність**
	Основна	**Другорядні**	
Контент	TEXT-ONLY	-	низька
Контент, Контекст	NON-GEO		середня
	ALL		середня
	TEXT-ONLY	USER-ONLY, GEO-ONLY	низька
	NON-GEO	**GEO-ONLY**	**висока**

Під час роботи було досліджено обмежену кількість комбінацій ознак тексту, як описано в Таблиці 2.2. На практиці кожна модель мала або один (тільки KF), або два (KF і MF) шари обгортки, які використовувалися відповідно до типу вхідних даних на етапі навчання. Зауважте, що кожна навчальна вибірка мала одну (тільки KF) або більше (KF і MF) токенізованих послідовностей, призначених їй у завантажувачі даних моделі, таким чином, що втрати на кожному кроці навчання залежали від одного (тільки KF) або більше (KF і MF) компонентів прогнозу та значень їх втрат. Ключова змінна (KF) використовувалася для валідації в кінці кожної епохи навчання, тоді як другорядні змінні (MF) впливали в першу чергу на втрати під час навчання.

Для досягнення більшої точності тестовий набір даних слід формувати відповідно до KF моделі, але без урахування модельних MF, оскільки шар обгортки MF слід використовувати лише під час навчання моделей. Найкраща комбінація вхідних змінних моделі, згідно з метриками продуктивності, NON-GEO як KF та GEO-ONLY як MF, що перевершили всі інші моделі GMOP з таким самим типом виходу.

Останній етап підготовки даних для завантаження до моделі полягав у перетворенні текстових характеристик набору даних у завантажувач даних у вигляді числових тензорів з відповідними ідентифікаторами, масками уваги та мітками координат. Цей процес розпочинається з фільтрації URL-адрес і безладної пунктуації з тексту, щоб усунути нерелевантну інформацію. Потім очищений текст кодується за допомогою стандартного токенізатора BERT, перетворюється на завантажувачі даних із заданим розміром пакету (batch) та готується до використання моделлю. Кожна вибірка в пакеті завантажувача даних представляє собою один твіт, закодований в ідентифікатори та маски уваги його текстових особливостей, а також позначений парою географічних координат. На практиці кількість слів у токенізованому текстовому корпусі не перевищувала 300, що відповідало розміру вхідних даних базової BERT-моделі 512 токенів (слів).

Опис послідовності обробки датасету твітів при навчанні моделі:

– розробник вводить параметри для ініціації моделі та назву файлу датасету для навчання;

– розробник запускає навчання моделі;

– розробник відстежує прогрес навчання моделі за консольним виводом метрик оцінки геопросторової та ймовірнісної точності;

– якщо процес навчання завершується успішно, то фінальний файл навченої моделі зберігається у відповідну папку.

Потрібно зазначити, що процес оцінки моделей схожий до процесу навчання моделей, але на початку ще завантажується локальний файл навченої моделі, а по завершенню оцінки зберігається файл обробленого датасету з

чисельними колонками результатів та файли статистики результатів (зображення графіків, текстові записи метрик).

По завершенню ітерацій навчання та оцінки моделей було отримано готову фінальну версію найкращої моделі яку було використано за основу для роботи Телеграм боту.

Опис послідовності обробки тексту користувача при прогнозуванні місцезнаходження використовуючи готову модель розміщену на HuggingFace:

– користувач запускає Телеграм бот;

– користувач вводить текст як аргумент команди «/predict»;

– текст користувача фільтрується на предмет пунктуації та посилань, токенізується стандартними методами BERT;

– якщо довжина токенізованого тексту користувача не менше 1 і не перевищує 512 токенів, то він подається на обробку у модель, інакше користувач бачить повідомлення про неправильно введений аргумент;

– якщо валідація тексту відбулась успішно, то через кілька секунд користувач отримує графічне та чисельно-текстове представлення результату прогнозу місцезнаходження.

2.2 Архітектура програмного забезпечення

Для побудови ПЗ було використано модульний тип архітектури зображений у графічному матеріалі «Схема структурна компонентів програмного забезпечення» як Balckboard шаблон архітектури. Цей патерн корисний для задач, для яких не відомі детерміновані стратегії розв'язання. Шаблон дошки складається з 3 основних компонентів:

– дошка − структурована глобальна пам'ять, що містить об'єкти з простору розв'язків (інтерфейси)

– джерело знань - спеціалізовані модулі з власним представленням (БД, результуючі файли)

– компонент управління − вибирає, конфігурує та виконує модулі (алгоритми навчання, оцінки, прогнозування).

Всі компоненти мають доступ до дошки. Компоненти можуть створювати нові об'єкти даних, які відображаються на дошці. Компоненти шукають певні типи даних на дошці і можуть знаходити їх за шаблоном, що співпадає з існуючим джерелом знань.

З огляду на те що розроблене ПЗ не передбачає використання БД під час навчання та оцінки моделей, діаграма сутностей на Рисунку 2.1 показує відношення між твітом, автором, геотегом, та прогнозом моделі що зберігаються у єдиному файлі табличного формату. Зокрема “user” та “place” вказують на поля JSON об’єкту твіта за якими формується датасет, а “GMM” вказує на формат результату прогнозу місцезнаходження.

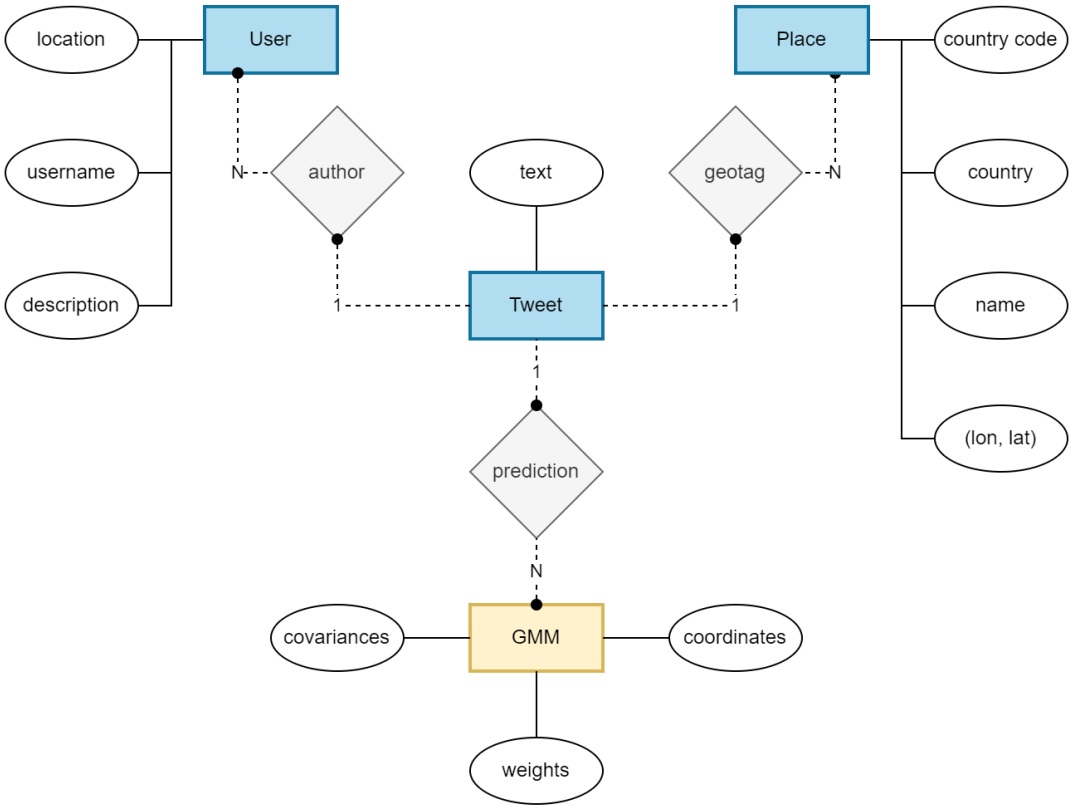


Рисунок 2.1 – ER діаграма сутностей Tweet, User, Place, GMM у датасетах з результатами оцінки

Потрібно зауважити що GMM найкращої моделі має сферичну матрицю коваріації, вагу, широту та довготу для кожного з 5 піків розподілу. Блакитним позначено сутності що мають вхідні дані моделі, а жовтим – вихідні.

Натомість для користувацького інтерфейсу у вигляді Телеграм боту було розроблено мінімальну БД для запису повідомлень від користувачів та відповідей на них. На Рисунку 2.2 зображено таблицю БД логів та її атрибути, де жирним наведено унікальний індекс, а пунктиром позначено колонки що можуть приймати нульове значення.

Усього існує три варіанти команд: start, info, predict. Текст відповіді на перші дві команди заноситься у БД в скороченому вигляді що потрібно для відстеження загальної кількості запитів до Телеграм боту. Текст відповіді на команду predict передбачає повідомлення про помилку (порожній аргумент тексту або задовгий текст) та результат прогнозу місцезнаходження у текстово-чисельному форматі. Текстовий аргумент зберігає вхідний текст наданий користувачем з відповідним тегом та іменем. Нарешті ID оновлення отримується з виклику Telegram API та є унікальним для кожного повідомлення що надходить у бота.

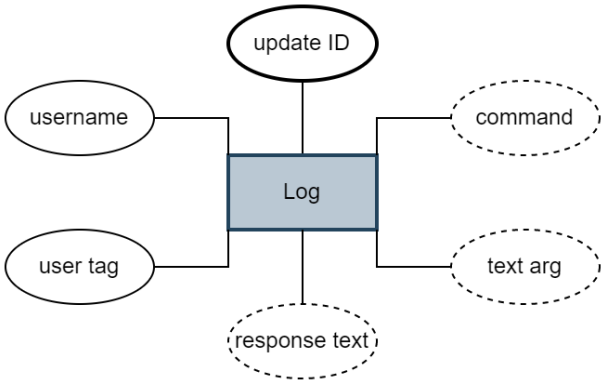


Рисунок 2.3 – ER діаграма сутності Log у БД Телеграм бота

2.3 Конструювання програмного забезпечення

У цій роботі зазначається, що багатомовній базовій моделі BERT було доручено виконати регресію для прогнозування числових значень, таких як значення координат, ваг та коваріацій GMM. Це завдання схоже на класифікацію тексту і вимагає модифікації останнього шару моделі. Базова BERT-модель має прихований розмір 768, який також є розміром токену прихованого стану для класифікації послідовностей. Запропонований шар-

обгортка працює, використовуючи лише вихід пулера BERT-моделі, який складається з оброблених токенів класифікації, кожен з яких представляє окремий твіт у пакеті. Обгортковий шар реалізує загальну логіку лінійної регресії, перетворюючи вектор розміром 768 у вихідний вектор заданого параметрами розміру. Точна кількість виходів залежить від типу моделі, що використовується. Крім того, всі моделі можна згрупувати в 4 типи за різницею в структурі їхніх виходів, як показано в Таблиці 2.3, де результати прогнозу визначають кількість прогнозованих географічних точок, Ключова та Другорядна змінні визначають шари обгортки запропонованих моделей, а код показує відповідні абревіатури моделей.

Таблиця 2.3 – Типи моделей за кількістю результатів, згруповані за просторовою та імовірнісною формами результатів

Тип	**Результати прогнозу**	**Ключова змінна**			**Другорядні змінні**		**Код**
		Точка	Вага	Ковар	Точка	Ковар	
Geo	1	**2**	-	-	**2**	-	GSOP
	M>1	**M*2**	**M**	-			GMOP
Prob	1	**2**	-	**1**	**2**	**1**	PSOP
	M>1	**M*2**	**M**	**M**			PMOP

Шар обгортки реалізовував лінійну регресію з динамічною кількістю виходів, яка залежала від типу змінної (ключова чи другорядна), типу моделі (геопросторова чи ймовірнісна) та кількості результатів прогнозування (один чи декілька). У найпростішому випадку прогнозування одного результату (SOP) ключова відмінність між геопросторовими та імовірнісними моделями полягала у формі виходу, який міг бути двовимірною точкою або двовимірним нормальним розподілом.

$$\hat{\mathbf{Y}}_{\mathbf{spat}} = (\hat{y}_{lon}, \hat{y}_{lat}); \hat{y}_{lon} \in \mathbb{R}; \hat{y}_{lat} \in \mathbb{R}$$

$$\hat{Y}_{prob} = N(\hat{\boldsymbol{\mu}}, \boldsymbol{\Sigma}); \hat{\boldsymbol{\mu}} = \hat{\mathbf{Y}}_{\mathbf{spat}}; \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{\hat{c}} & 0 \\ 0 & \sigma_{\hat{c}} \end{bmatrix}; \sigma_{\hat{c}} > 0$$

Імовірнісний вихід моделі включав не тільки просторову складову прогнозованої пари координат $\hat{\mu}$, але й міру впевненості моделі у своєму прогнозі. Матриця коваріацій Гауса, яка представляє невизначеність моделі,

була сферичного типу і могла бути визначена єдиним додатним ненульовим числовим значенням σ. Для забезпечення того, щоб $\sigma_{\hat{c}}$ залишалася додатною, функція SoftPlus, описана в рівнянні (SoftPlus), застосовується до вихідної змінної $\hat{c}$.

$$\sigma_{\hat{c}} = \log(1 + e^{\hat{c}}); \hat{c} \in \mathbb{R}; \sigma_{\hat{c}} > 0 \text{ (SoftPlus)}$$

Однак нижня межа для $\sigma_{\hat{c}}$ була встановлена на рівні $\frac{1}{2\pi}$ для збереження значення PDF прогнозованих GMM, що описується рівнянням (PDF), в діапазоні [0, 1].

$$\sigma_{\hat{c}} = \log(1 + e^{\hat{c}}) + \frac{1}{2\pi}; \hat{c} \in \mathbb{R}; \sigma_{\hat{c}} \in (\frac{1}{2\pi}, +\infty) \quad \text{(LBSP)}$$

Вибір сферичної матриці коваріації ґрунтується на емпіричних даних, які свідчать про менші геопросторові помилки порівняно з моделями, що використовують діагональну, повну та зв'язану коваріаційні матриці. Хоча більш складні матриці, такі як діагональні та повні, надають більше свободи у формі розподілу, вони вимагають більшої кількості вихідних даних. Вибір мінімальної кількості значень для визначення форми гаусівського розподілу усунув ризик того, що модель надасть перевагу оптимізації ймовірнісної складової втрат, описаної в рівнянні (NLLH), над просторовою складовою втрат, описаною в рівнянні (SED).

Крім того, встановлена нижня межа $\frac{1}{2\pi}$ для $\sigma_{\hat{c}}$ зменшила гостроту гауссових піків і забезпечила їхню висоту ніколи не перевищуючу 1, таким чином уникнувши від'ємних значень ймовірнісних втрат, як показано на Рисунку 2.5. В результаті застосування нижньої межі SoftPlus, описаної в рівнянні (LBSP), до виходів, пов'язаних з параметром коваріації, як L_{spat}, так і L_{prob}, залишалися в додатній області і наближалися до 0 під час навчання моделі. Таким чином, це усунуло різницю в імпульсі втрат і дозволило включити як геопросторові, так і ймовірнісні помилки як рівнозначні компоненти втрат під час обчислення загальних втрат пакету.

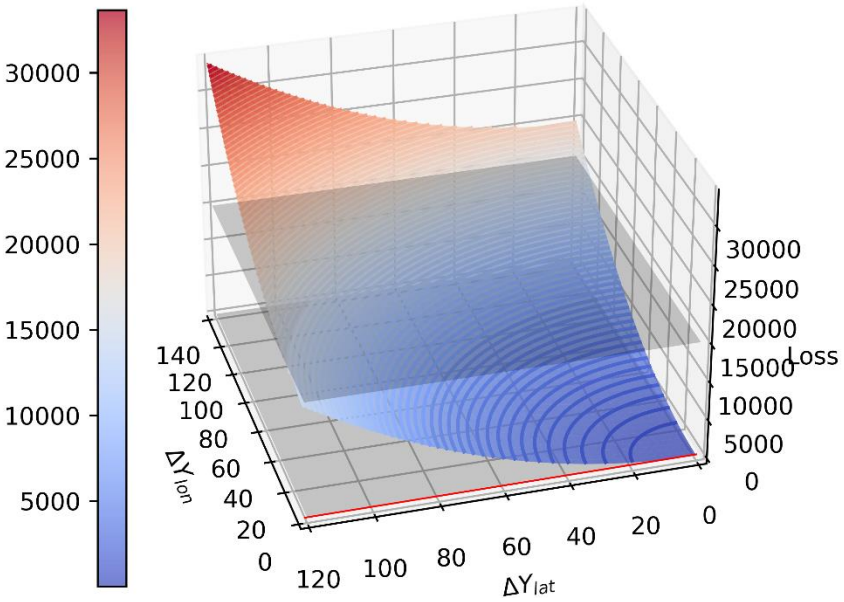


Рисунок 2.4 – Поверхня функції квадратичної евклідової відстані (SED) на осях ΔY_{lon} та ΔY_{lat} як абсолютних відстанях похибки по осях довготи та широти; верхня горизонтальна сіра поверхня вказує на емпіричний максимум L_{spat}; червона лінія вказує на строгий мінімум 0, що випливає з природи рівняння (SED).

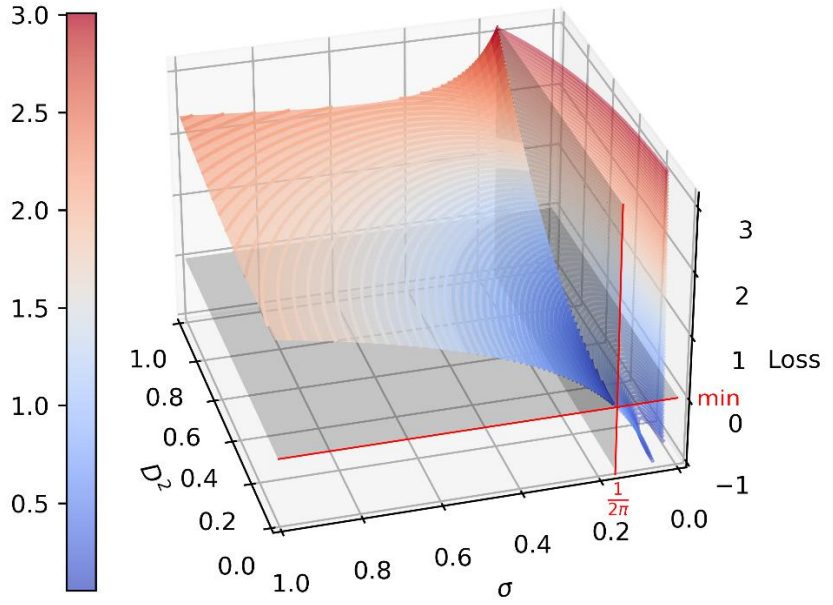


Рисунок 2.5 – Поверхня функції (NLLH) на осях D^2 як квадрату відстані помилки та $\sigma_{\hat{c}}$ як невизначеність у $\hat{\mu}$ GMM; червоні лінії та сірі поверхні вказують на зменшення області L_{prob} в результаті застосування рівняння (LBSP)

Щоб врахувати варіації вихідних змінних, кожен тип моделі вимагав індивідуального методу обчислення втрат. Так, моделі типу "Геопросторові" використовували лише L_{spat}, в той час як моделі імовірнісного типу використовували комбінацію L_{spat}та L_{prob}. З точки зору оцінки виходу KF, процедури обчислення втрат для моделей типу SOP та MOP візуалізовано як обчислювальні графи у графічному матеріалі «Схема структурна станів програмного забезпечення».. Повна структура найкращої із запропонованих моделей, а саме PMOP з використанням декількох текстових ознак, також показана у вигляді графу обчислення втрат у графічному матеріалі «Схема структурна станів програмного забезпечення».

Просторові втрати для моделей типу SOP розраховувалися як квадрат евклідової відстані між справжнім місцезнаходженням, вказаним користувачем, і місцезнаходженням, передбаченим моделлю на двовимірній проекції Меркатора. Зауважте, що L_{spat} за своєю природою завжди залишається більшою або дорівнює нулю, як показано на Рисунку 2.4.

$$L_{GSOP} = (\mathbf{Y} - \widehat{\mathbf{Y}})^2 = (y_{lon} - \hat{y}_{lon})^2 + (y_{lat} - \hat{y}_{lat})^2 = D^2;\ D \geq 0 \text{ (SED)}$$

Імовірнісна складова втрат моделей PSOP розраховувалася як від'ємна log ймовірність того, що вихідна точка Y впишеться в прогнозований розподіл N ($\hat{\mu}$, Σ) як описано в рівнянні (NLLH).

$$PDF = N(\mathbf{Y} \mid \widehat{\boldsymbol{\mu}}, \boldsymbol{\Sigma}) = \frac{e^{-\frac{D^2}{2\sigma}}}{2\pi\sigma};\ PDF \in [0,1] \text{ (PDF)}$$

$$L_{PSOP} = -\log(PDF) = \frac{D^2}{2\sigma} + \log(2\pi\sigma);\ L_{PSOP} \geq 0 \text{ (NLLH)}$$

Важливо зазначити, що визначення гауссової матриці коваріації передбачає, що σ має бути більшим за 0, що може бути досягнуто застосуванням рівняння (SoftPlus) з нижньою межею 0. Однак, припускаючи, що відстань помилки наближається до 0, невизначеність моделі в прогнозованій точці також наближатиметься до 0. Як наслідок, рівняння (PDF) перевищить 1, що призведе до від'ємних значень рівняння (NLLH):

$$\lim_{(\sigma,D)\to(0^+,0^+)} PDF = +\infty; \quad \lim_{(\sigma,D)\to(0^+,0^+)} -\log(PDF) = -\infty$$

Це дослідження пропонує застосування рівняння (LBSP) для обмеження параметра коваріації, як показано на Рисунку 2.6. Отже, коли міра невизначеності σ наближається до свого мінімуму при $\frac{1}{2\pi}$, L_{prob} покладається в основному на її просторову складову D^2:

$$\lim_{\sigma \to \frac{1}{2\pi}} PDF = e^{-\pi D^2};\ \lim_{\sigma \to \frac{1}{2\pi}} L_{PSOP} = \pi D^2;$$

У випадку моделей типу MOP вихідні дані включали вагу $\widehat{w}_i$. для кожного з M виходів що свідчить про його значущість серед інших результатів. Щоб переконатися, що всі прогнозовані ваги в сумі дорівнюють 1, до $\widehat{w}_i$ було застосовано функцію SoftMax, описану в рівнянні (SoftMax).

$$W_i = \frac{e^{\widehat{w}_i}}{\sum_{j=1}^{M} e^{\widehat{w}_j}};\ \widehat{w} \in \mathbb{R};\ \sum_{i=1}^{M} W_i = 1;\ W_i \in [0,1] \quad \text{(SoftMax)}$$

Тому сумарні геопросторові та імовірнісні втрати MOP були розраховані як зважена лінійна комбінація всіх помилок M результатів, описаних у рівнянні (WLC).

$$W_i = \frac{e^{\widehat{w}_i}}{\sum_{j=1}^{M} e^{\widehat{w}_j}};\ \widehat{w} \in \mathbb{R};\ \sum_{i=1}^{M} W_i = 1;\ W_i \in [0,1]$$

$$L_{GMOP} = \sum_{i=1}^{M} W_i D_i^2;\quad L_{PMOP} = \sum_{i=1}^{M} W_i \left(\frac{D_i^2}{2\sigma_i} + \log\left(2\pi\sigma_i\right)\right) \quad \text{(WLC)}$$

Сумарні втрати на кожну текстову змінну f мали стандартний геопросторовий компонент, тобто $L_f = L_{\text{spat}}$, і, за бажанням, до них додавалася ймовірнісна втрата. Втрати для імовірнісних моделей розраховувалися трьома способами: середнє значення або сума L_{spat} і L_{prob}, або L_{prob} без урахування L_{spat}. Перший варіант дещо перевершив другий варіант і значно перевершив останній варіант, тому втрати на одну змінну були розраховані наступним чином:

$$L_f = \frac{L_{spat} + L_{prob}}{2}$$

Нарешті, для обробки декількох текстових змінних F одного твіту було розраховано загальний збиток як середнє значення одного KF і K MF. Зауважте, що кількість результатів M була пов'язана виключно з результатами

KF, тоді як MF зберігали результати типу SOP. Однак кількість MF K дорівнювала F – 1 і могла бути будь-яким цілим числом, більшим або рівним нулю.

$$L_{total} = \frac{\sum_{i=1}^{F} L_i}{F}; F \geq 1$$

На практиці останнім кроком було обчислення середнього значення загальних втрат на один твіт в одному тензорі завантажувача даних для зворотного розповсюдження значення з плаваючою комою, що представляє загальну суму втрат на партію твітів. Експерименти також виявили зростання часу обчислень на 12% у випадку ймовірнісних моделей порівняно з геопросторовими аналогами. Наше обладнання (NVIDIA GeForce GTX 1080 Ti) змогло обробити геопросторові розрахунки втрат зі швидкістю 16,7 твітів на секунду (т/с) порівняно з 14,9 т/с для комбінованих геопросторових та імовірнісних розрахунків втрат, необхідних у випадку імовірнісних моделей.

Згідно з експериментами, найкращою була модель імовірнісного прогнозування багатьох результатів (PMOP) з 5 піків GMM, що використовує NON-GEO як ключову змінну (KF) і GEO-ONLY як другорядну змінну (MF), яка була навчена за допомогою спеціальних процедур обчислення втрат, описаних у графічному матеріалі «Схема структурна діяльності». Структурні схеми класів якого наведені у Додатку Б.

Навчання та оцінка моделей на датасетах Твітера запускається з єдиної точки входу що відповідає за параметризацію моделі та зчитування файлу датасету для формування завантажника даних моделі. Під час навчання моделі використовуються функції обчислення втрат для чисельних даних у тензорному форматі, що дозволяє навчати моделі на базі GPU за допомогою бібліотеки PyTorch та її модуля для роботи з чисельними розподілами.

Після завершення оцінки моделі формуються колонки результуючого датасету з прогнозами місцезнаходження у вигляді параметрів GMM що передбачає пост обробку вихідних числових значень моделі. На базі отриманих результатів для датасету можливо вирахувати статистику помилок

та оцінити роботу моделі за метриками продуктивності, а також отримати графічне представлення результатів. Виклик методів для обробки виходу моделі та візуалізації результатів відбувається після закінчення процесу оцінки датасету або з окремої точки запуску що дозволяє аналізувати вже збережені файли з прогнозами місцезнаходження в залежності від параметрів моделі. Для прогнозування окремих текстів з використанням локально збереженої або завантаженої з HuggingFace моделі також існує окрема точка входу, що використовує вже існуючі класи та функції.

Під час навчання та оцінки моделі використовується обгортковий клас самої моделі, що задає останній шар її архітектури на базі BERT моделі. Такий обгортковий шар ініціалізується при параметризації моделі користувачем та зберігає ключові ознаки моделі, зокрема розмітку вихідних параметрів кожної текстової змінної. Регрессор реалізує обгорткові шари для KF та MF за допомогою лінійного перетворення 768 векторів у задану параметрами моделі кількість числових виходів.

Для реалізації користувацького інтерфейсу у вигляді Телеграм боту запроваджено обгортковий клас для чату що має доступ до мінімального набору класів та функцій потрібних для обробки окремих текстових змінних та до БД логів. За можливості, такий бот використовує скорочені версії класів описаних вище для роботи з результатом моделі, а використання моделі регресії зі стандартними методами бібліотеки HuggingFace реалізовано методами NumPy завдяки базовому класу Pipeline.

В якості системи управління базами даних використовується SQLite. База даних серверу призначена для зберігання повідомлень від користувачів та відповідей на них. Опис таблиці БД користувацького інтерфейсу у вигляді Телеграм боту наведено у Таблиці 2.4

Таблиця 2.4 – Опис таблиці Logs

Таблиця	Назва поля	Тип даних	Опис
LOGS	ID	INT	Ідентифікаційний номер повідомлення користувача
	USER	TEXT	Ім’я користувача
	TAG	TEXT	Телеграм тег користувача
	CMD	TEXT	Команда з повідомлення користувача
	TEXT	TEXT	Текст або аргумент команди з повідомлення користувача
	OUT	TEXT	Текст відповіді бота на повідомлення користувача

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.5.

Таблиця 2.5 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	PyCharm	Головне середовище розробки ПЗ дипломної роботи на мові Python.
2	Docker	Відкрита платформа для розробки, доставки та запуску програм, що спрощує процес створення, запуску, керування та розповсюдження додатків шляхом віртуалізації операційної системи комп’ютера, на якому він встановлений і працює, для розгортання серверної частини Телеграм боту.

Продовження Таблиці 2.5

3	SQLite	Програмна бібліотека, яка забезпечує автономну, безсерверну, транзакційну СУБД SQL у вигляді єдиного файлу що містить усі таблиці та індекси БД для збереження історії повідомлень Телеграм боту.
4	Slurm Workload Manager	Планувальник завдань із відкритим вихідним кодом для Linux і Unix-подібних ядер, який використовується багатьма світовими суперкомп'ютерами та комп'ютерними кластерами, для паралельного тренування моделей на сервері з GPU.
5	HuggingFace	Платформа від спільноти штучного інтелекту яка надає інструменти для побудови, навчання та розгортання моделей машинного навчання на основі відкритого коду та сучасних технологій.
6	transformers	Python бібліотека що надає API та інструменти для легкого завантаження та навчання найсучасніших попередньо навчених NLP моделей з HuggingFace
7	TensorBoard	Інструмент для забезпечення вимірювань та візуалізації, необхідних під час процесу машинного навчання.
8	PyTorch	Python фреймворк машинного навчання з відкритим вихідним кодом, використовується для розробки та навчання моделей глибокого навчання на основі нейронних мереж.
9	Pandas	Python бібліотека що надає різні структури даних і функції для легкої та інтуїтивно зрозумілої роботи з датасетами.

Продовження Таблиці 2.5

10	Matplotlib, Seaborn	Python бібліотеки які дозволяє користувачам створювати статичні, анімовані та інтерактивні візуалізації для графічного відображення результатів.
11	Geopy	Python бібліотека що спрощує обчислення географічних відстаней між двома точками для вирахування відстані похибки у кілометрах.
12	NumPy, SciPy	Python модулі з відкритим вихідним кодом що надають загальні математичні та числові процедури у вигляді попередньо скомпільованих швидких функцій для аналізу числових даних.
13	requests	Python бібліотека що дозволяє викликати Telegram API з метою отримання та відправки повідомлень у чат для реалізації користувацького інтерфейсу прогнозування у вигляді Телеграм боту.
14	GPutil, psutil	Python модулі GPutil та psutil що надають засоби для аналізу використання ресурсів машини для відстеження навантаження на GPU та CPU.
15	Basemap	Python модуль що дозволяє генерувати світову мапу різної деталізації для візуалізації результатів прогнозування на проекції Меркатора.
16	Scikit-learn	Python бібліотека аналізу даних з відкритим кодом і бібліотека машинного навчання що надає вибір ефективних інструментів для машинного навчання та статистичного моделювання.
17	argparse	Python модуль який полегшує написання зручних інтерфейсів командного рядка, а саме для швидкої параметризації запуску навчання та оцінки моделей.

2.4 Аналіз безпеки даних

У датасетах для навчання та оцінки присутні дані що містять інформацію про користувача, що не повинні бути оприлюднені за політикою Твітера. Тож зразки датасетів не можуть бути надані у повному об’ємі разом з відкритим вихідним кодом ПЗ.

При використанні Телеграм боту відбувається логування повідомлень користувачів у БД де зазначається тег, ім’я та текст повідомлення користувача, що теж потребує обмежень у доступі з боку інших користувачів. Тому повідомлення від користувачів фільтруються від знаків пунктуації перед обробкою в моделі та логуванням у БД для уникнення ін’єкцій коду.

Висновки до розділу

Було описано модульну архітектуру ПЗ та методологію машинного навчання моделі нейронної мережі на базі BERT що реалізує задачу регресії до числових параметрів GMM як форми місцезнаходження на мапі Меркатора.

Ця робота була спрямована на вивчення методів машинного навчання для розв'язання задачі прогнозування геолокації коротких текстів за допомогою методів NLP із застосуванням нейронних мереж на основі BERT. Запропонована модифікація архітектури моделі виграє завдяки простоті налаштування, необхідного для навчання моделей під конкретну задачу користувача. Моделі є гнучкими до будь-якого рівня геопросторової деталізації (світ, країна, місто і т.д.) шляхом зміни набору даних для навчання та мовної специфіки базової попередньо навченої BERT-моделі.

Найефективніші моделі були навчені за багатозадачним алгоритмом що надає можливість паралельного навчання із загальними втратами на пакет твітів. Найкращою стратегією було розділення на ключові NON-GEO (завжди присутні "текст" і "користувач") і другорядні GEO-ONLY ("місце" присутнє тільки в твітах з геотегами) текстові ознаки. На практиці, кожна ознака мала свій окремий шар лінійної регресії, що обгортав базову модель, які

відрізнялися за кількістю результатів прогнозування, але збігалися за типом функції втрат для кожної вхідної змінної (геопросторова/імовірнісна). Оцінювання проводилося з використанням лише шару обгортки ключової змінної без урахування шару обгортки другорядних змінних, який використовувався лише під час навчання моделі. Такий підхід дозволив зберегти формат загальних вхідних даних моделі та зіставити автоматично згенеровані описи місць з локаціями та текстом вказаними користувачами. З точки зору описаних налаштувань, найкращі результати були досягнуті при використанні другорядних втрат типу SOP та ключових втрат типу MOP для 3-5 результатів прогнозування.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Оскільки порівняння моделей для прогнозування місцезнаходження відбувається на базі загальнодоступних датасетів та метрик, то далі буде наведено порівняння минулих робіт та запропонованих моделей.

Існує три реальні набори даних Twitter, які широко використовувалися в попередніх роботах для оцінки моделей геолокації:

GeoText [10] – це відносно невеликий набір даних Twitter, що складається з 9,5 тис. користувачів тільки з США. Цей набір даних призначений для оцінки моделей, що розв'язують задачу прогнозування домашнього місцезнаходження користувача, і складається з твітів без мета-даних про користувача або місце, придатних для гібридного (контент та контекст) підходу, запропонованого в цій роботі.

Twitter-US [30] – це більший набір даних, що складається з 449 тис. користувачів зі США, цей набір даних також згадується як UTGeo2011 в деяких роботах [7], [30]. Подібний набір даних твітів з геотегами лише з США було зібрано з нашого архіву бази даних Twitter як порівнянний аналог, який містить як текстові дані, так і метадані.

Twitter-World [12] – це набагато більший набір даних, який складається з 1,3 мільйона користувачів з різних країн світу. Основне місцезнаходження користувачів прив'язане до географічного центру міста, звідки публікується більшість їхніх твітів. Аналогічно, з нашого архіву було зібрано всесвітню базу даних твітів з геотегами, щоб виконати прогнозування геолокації на основі твітів за допомогою текстового та гібридного (контент та контекст) підходів.

Існувало кілька причин для створення альтернативних тестових наборів даних Twitter-US та Twitter-World для оцінки моделі як показано у Додатку Г. По-перше, на даний момент не було прямого доступу до оригінальних наборів

даних. По-друге, ми мали доступ до локального архіву Твіттера, який містив усі твіти з 2020 по 2022 рік, що було зручніше для збору та аналізу великих масивів даних. Нарешті, це дозволило нам скласти орієнтовані на завдання набори даних, які мають специфічні текстові особливості, такі як контекстні метадані "користувача" і "місця". Оскільки вихідні набори даних в основному призначені для завдання прогнозування геолокації користувача, порівняння з метриками попередніх робіт у Таблиці 3.1 є приблизним і слугує лише видимим орієнтиром для найкращої із запропонованих модифікацій BERT-моделі.

Важливо, що тестова оцінка для локально відтворених Twitter-US і Twitter-World була проведена на наборах даних, відфільтрованих від користувачів-ботів (які публікують більше 20 повідомлень на день) і користувачів, використаних у навчальних наборах даних, щоб гарантувати, що валідація була неупередженою і забезпечила точну оцінку узагальнюючих здібностей моделей.

Метрики ефективності поділяються на геопросторові та імовірнісні, перші є універсальними для всіх моделей, а другі застосовуються лише до імовірнісних моделей. Опис імовірнісних метрик наведено у Додатку Д. Далі наведено застосовані геопросторові метрики:

– **SAE** – simple accuracy error – проста метрика похибки точності використовується для вимірювання географічної відстані між двома точками на поверхні Землі. Це досягається шляхом застосування формули гаверсинуса для обчислення відстані по великому колу в кілометрах:

$$SAE_{SOP} = Hav(\mathbf{Y}, \widehat{\mathbf{Y}}) = D_H$$

де Y та $\hat{Y}$ представляють оригінальні та прогнозовані точки відповідно. При обчисленні метрик моделі типу MOP, SAE обчислюється як зважена лінійна комбінація всіх M результатів GMM прогнозу моделі:

$$SAE_{MOP} = \sum_{i=1}^{M} W_i Hav(\mathbf{Y}, \widehat{\mathbf{Y}}_i) = D_H$$

Середнє та медіана SAE для валідаційного набору даних використовуються як ключові показники якості. Варто зазначити, що під час процесу навчання моделей розрахунок просторової похибки спрощується до формули евклідової відстані.

– **Acc@161** – відсоток прогнозованих місць, які знаходяться в радіусі 161 км (100 миль) від фактичного місцезнаходження твіта:

$$Acc@161 = \frac{\left|i: D_{H,i} \leq 161\right|}{N} \cdot 100$$

де D_H – множина з N елементів, що представляє відстань по великому колу між прогнозованими та істинними місцеположеннями, та |i : $D_{H,i} \leq$ 161| – кількість елементів у D_H, відстань між якими менша або дорівнює 161 км. Зауважимо, що ця метрика, заснована на імперських одиницях, стала однією з найпопулярніших у сфері прогнозування місцезнаходження завдяки великому внеску дослідників зі Сполучених Штатів.

Загалом, для порівняння з попередніми роботами в Таблиці 3.1 використано лише геопросторові метрики, такі як Acc@161 (у відсотках), середнє та медіанне значення SAE (у кілометрах). Зауважте, що обидва набори даних Twitter-US і Twitter-World були відтворені локально з архівної бази даних постів Twitter за 2021 рік.

Геопросторові метрики в Таблиці 3.1 демонструють застосування найкращої моделі до текстових даних (текст і метадані) твітів у трьох наборах даних, описаних вище. Таблиця розділена на блоки аналізу прогнозування домашнього місцезнаходження користувача і місцезнаходження твіту через поширеність першого завдання в суміжних роботах і відсутність результатів для другого завдання. Хоча основною метою цього дослідження було покращення точності прогнозування місцезнаходження твітів для даних світового масштабу, узагальнення прогнозів твітів для домашнього місцезнаходження користувача показує меншу відстань помилки і перевершує попередні роботи за середнім значенням SAE і Acc@161 на наборі даних Twitter-World. Статистика набору даних Twitter-US також вказує на кращі

показники середнього SAE та Acc@161. На противагу цьому, набір даних GeoText не має належних метаданих, що призвело до дуже високих значень похибки.

Таблиця 3.1 – Порівняння запропонованих моделей та суміжних робіт за результатами геопросторових метрик прогнозування місцезнаходження твіту та домашнього місцезнаходження користувача; ЗВ – змінна валідації визначає вхідні змінні оцінювання: Т – текстовий вміст, М – контекст метаданих, Н – гібрид тексту та мережі користувачів

Модель	**ЗВ**	**GeoText**			**Twitter-US**			**Twitter-World**		
		Mean SAE	**Med SAE**	**Acc 161**	**Mean SAE**	**Med SAE**	**Acc 161**	**Mean SAE**	**Med SAE**	**Acc 161**
Завдання пошуку домашнього місцезнаходження користувача										
Eisenstein [10]	T	845	501	-		-			-	
Eisenstein [9]	T	900	494	-		-			-	
Wing [38]	T	967	479	-		-			-	
Wing [39]	T	808	317	41	704	171	49	1715	490	33
Roller [30]	T	897	432	36	860	463	35		-	
Han [13]	TM		-		814	260	45	1953	646	24
Cha [3]	T	581	425	-		-			-	
Melo [23]	T		-		702	208	-		-	
Rahimi [29]	T	880	397	38	687	159	50	1724	530	32
	H	654	151	50	620	157	50		-	
Rahimi [28]	H	578	61	59	515	77	61	1280	104	53
		581	57	59	529	78	60	1403	111	53
	T	844	389	38	554	120	54	1456	415	34
		856	360	40	581	91	55	1417	373	36
Miura [24]	H		-		336	42	70	780	-	72
Do [7]	H	570	58	59	474	157	51		-	
Ebrahimi [8]	H	476	32	**64**	438	56	66	1216	95	54
Miyazaki [25]	T	821	325	44		-			-	
Huang [14]	T		-		455	64	61	762	86	56
	H		-		323	28	73	610	**6**	68
Zhong [45]	T	834	403	41	544	120	54	1456	415	34
	H	514	38	62	452	67	63	1107	102	55
Zheng [43]	H	516	30	64	359	31	72	818	49	62
Zhou [46]	H	**316**	**23**	-	**263**	37	-	636	62	-
Запропонована	T	1433	743	0	431	15	78	892	31	74
	TM	1059	741	1	375	**13**	**83**	**567**	26	**82**
Завдання пошуку місцезнаходження твіта										
Priedhorsky [26]	T	870	534	-		-			-	
		954	493							
Hulden [15]	T	765	357	-		-			-	
Liu [21]	T	856	-		733	377	-		-	
Bakerman [2]	H	**593**	**19**	-		-			-	
Запропонована	T	1216	787	1	1163	599	41	1588	50	61
	TM	1094	770	1	802	25	64	800	25	80

Потрібно зауважити що процедури прогнозування домашнього місцезнаходження користувача базуються на множині результатів у форматі GMM для окремих твітів користувача. З множини усіх центрів GMM для одного користувача формується координатна сітка, для кожної точки якої

послідовно обраховується PDF для отримання середніх значень з усіх результуючих розподілів GMM. Після обчислення поверхні функції вірогідності для користувача, на ній визначається 5 локальних максимумів. Для обраних піків генеруються ваги на базі їх попередньо отриманих середніх значень що в сумі складають 1. Такий метод дозволив приблизно визначити домашнє місцезнаходження користувача у форматі зважених координатних пар та порівняти його з найпопулярнішою (або найдавнішою) локацією з підмножини твітів користувача. При цьому точність прогнозу трохи покращилась у порівнянні з прогнозуванням місцезнаходження окремих твітів як показано у Таблиці 3.1. Приклад прогнозу домашнього місцезнаходження користувача наведено у Додатку В.

Аналізуючи результати прогнозу місцезнаходження найкращої моделі для всесвітнього датасету, можна зробити висновок що з 5 заданих піків GMM тільки 1-2 мають значну вагу як показано на Рисунку 3.1.

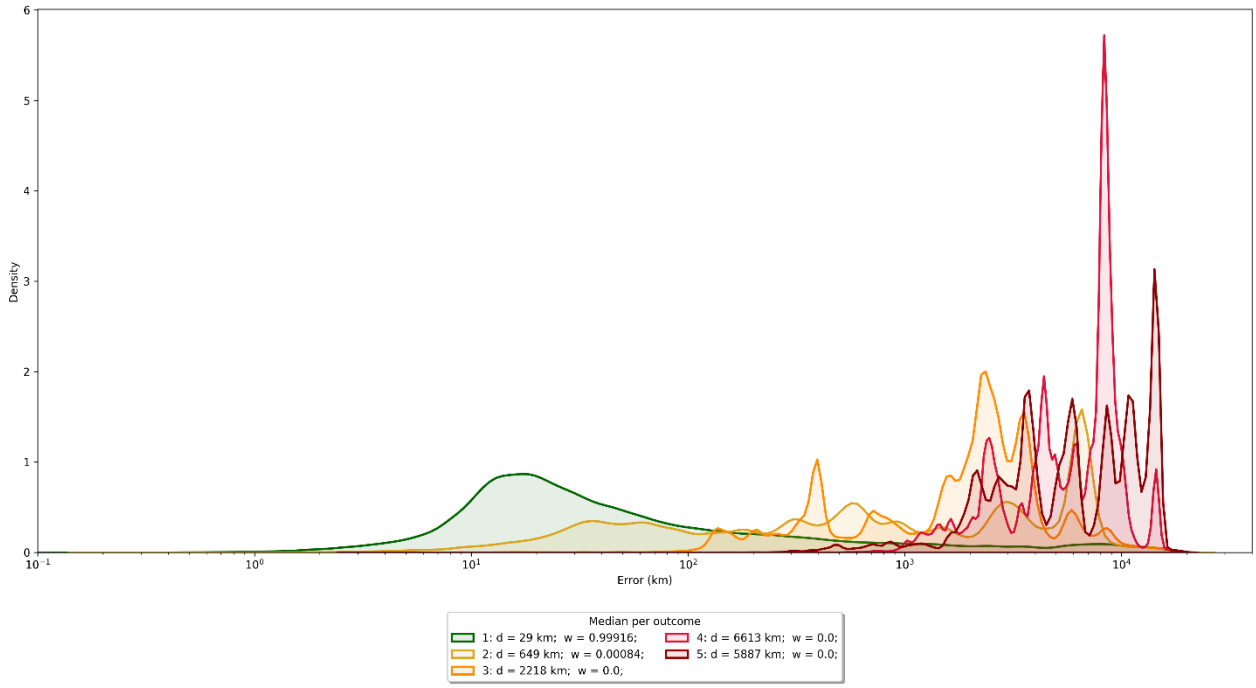


Рисунок 3.1 – Щільність логарифму відстані помилки в км на пік GMM для найкращої моделі розрахована на наборі даних 100,000 твітів з усього світу; медіани значень помилки та ваги на пік зазначені в легенді під графіком

Загалом, більше 70% результатів для най вірогіднішого піку GMM знаходяться в радіусі 100 миль від оригінальних точок як показано на Рисунку 3.2.

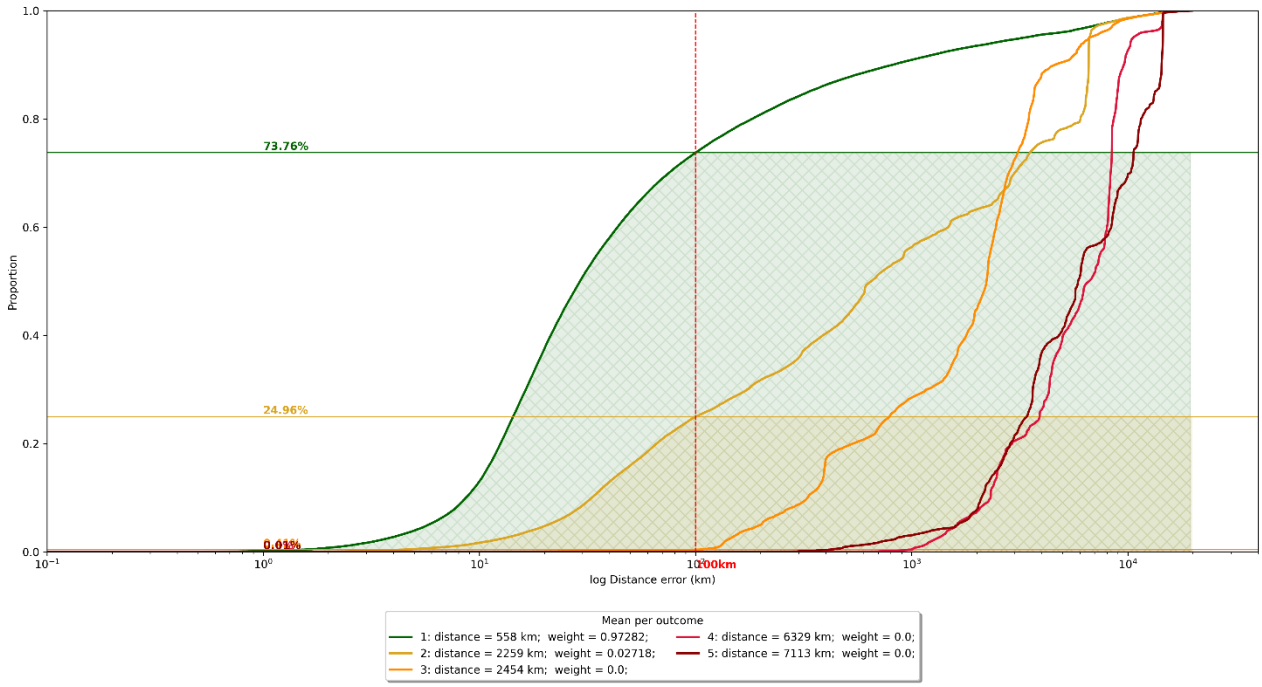


Рисунок 3.2 – Кумулятивний розподіл логарифму відстані помилки в км на пік GMM для найкращої моделі розрахована на наборі даних 100,000 твітів з усього світу; середні значення помилки та ваги на пік зазначені в легенді під графіком

Незважаючи на відносно малу кількість (2 з 5) значних точок результату, експерименти показали що зменшення (або збільшення) кількості піків GMM при навчанні не призводить до покращення точності прогнозів моделі. Більш того, моделі SOP показали себе гірше за MOP. В той же час використання метаданих в додаток до тексту твіта при оцінюванні моделі зменшує відстань похибки приблизно в 3 рази. Ефективність Геопросторових та Імовірнісних моделей мало відрізнялась, але останні показали трохи кращі результати. Таким чином найкраща модель була визначена як PMOP з 5 піків за типом прогнозу з використанням рівняння (LBSP) обмеження вихідного параметру коваріації GMM. Крім цього, комбінацію навчальних змінних KF NON-GEO та MF GEO-ONLY було визначено як найкращу з можливих. Порівняння

моделей всесвітнього рівня з різними налаштуваннями параметрів наведено у Додатку Г.

3.2 Опис процесів тестування

Було виконано статичне тестування всіх файлів тексту програми за допомогою сервісу rmbold як показано на Рисунку 3.3.

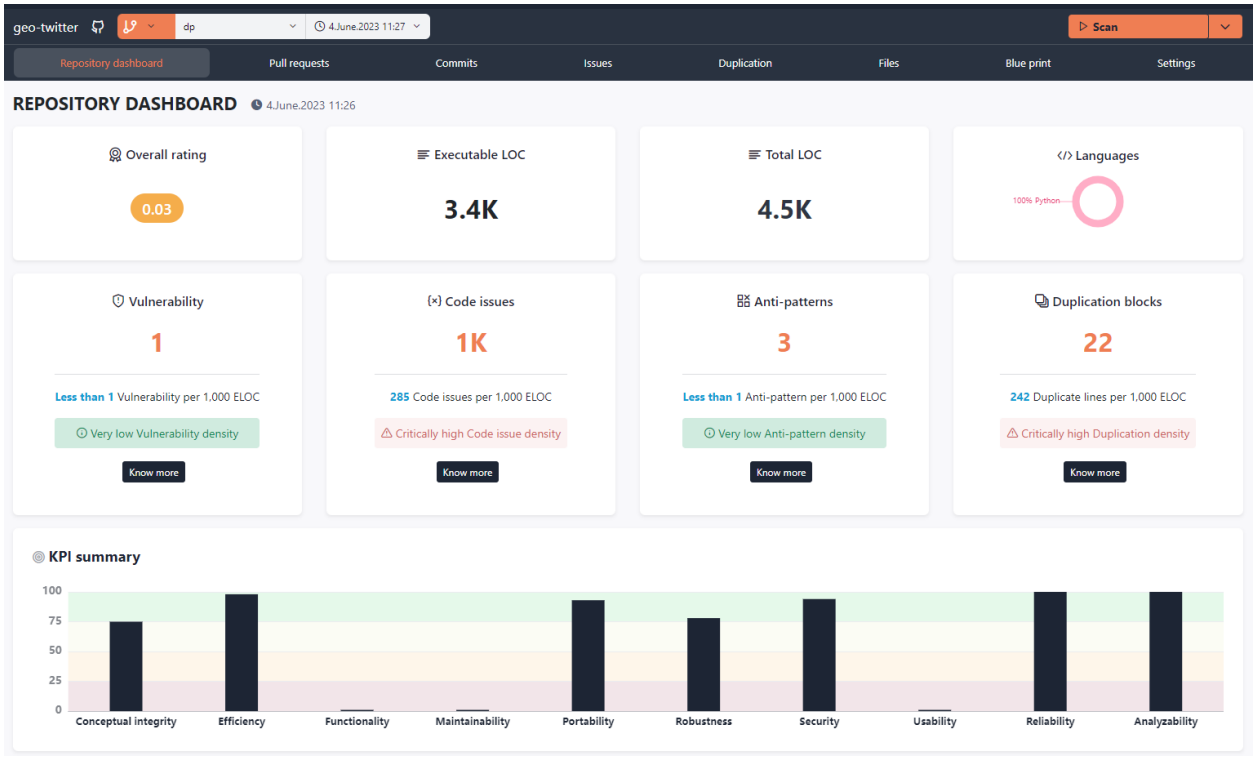


Рисунок 3.3 – Звіт статичного тесту embold

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у Таблицях 3.2 – 3.5.

Таблиця 3.2 – Тест 1.1

Тест	Навчання моделі на датасеті
Модуль	ПЗ Розробника
Номер тесту	1.1
Початковий стан системи	Розробник встановив ПЗ за інструкцією та відкрив консоль у кореневій папці проекту, у папці датасетів знаходиться відповідний файл навчальної послідовності твітів

Вхідні данні	Параметри навчання моделі, назва файлу датасету
Опис проведення тесту	Запускається навчання моделі, параметризація відбувається за допомогою прапорів з аргументами, або за допомогою редагування змінних у скрипті вхідної точки навчання та оцінки моделей. На початку відбувається ініціація обгортки моделі з заданими параметрами, файл датасету попередньо обробляється та токенізується для подальшого завантаження у модель. Під час навчання пакети вхідних даних подаються на вхід, на виході отримані числові змінні проходять пост обробку для формування параметрів GMM що використовуються для обчислення втрат при навчанні. Метрики точності записуються у папку логів TensorBoard та виводяться у консоль. В кінці кожної епохи відбувається оцінка моделі на тестовому датасеті (підмножина з 10% твітів навчального датасету) та збереження чекпоінту моделі у випадку зменшення тестової похибки. Наприкінці навчання відбувається збереження фінального файлу навченої моделі.
Очікуваний результат	Навчання запускається, проходить, та завершується успішно, Розробник отримує файл навченої на власному датасеті моделі, у папці логів навчання зберігається статистика прогресу навчання моделі
Фактичний результат	Навчання запускається, проходить, та завершується успішно, Розробник отримує файл навченої на власному датасеті моделі, у папці логів навчання зберігається статистика прогресу навчання моделі

Таблиця 3.3 – Тест 1.2

Тест	Оцінка моделі на датасеті
Модуль	ПЗ Розробника
Номер тесту	1.2
Початковий стан системи	Розробник встановив ПЗ за інструкцією та відкрив консоль у кореневій папці проекту, у папці датасетів знаходиться відповідний файл тестової послідовності твітів, у папці моделей знаходиться файл навченої моделі
Вхідні данні	Параметри оцінки моделі, назва файлу датасету, назва файлу моделі

Перенесення Таблиці 3.3

Опис проведення тесту	Запускається оцінка моделі, параметризація відбувається за допомогою прапорів з аргументами, або за допомогою редагування змінних у скрипті вхідної точки навчання та оцінки моделей. На початку відбувається ініціація обгортки моделі з заданими параметрами, завантаження навченої моделі з файлу у пам’ять, файл датасету попередньо обробляється та токенізується для подальшого завантаження у модель. Під час оцінки пакети вхідних даних подаються на вхід, на виході отримані числові змінні проходять пост обробку для формування параметрів GMM що записуються у новий масив даних тої ж довжини що і вхідний датасет. У консоль виводиться інформація про прогрес оцінки у вигляді метрик. Наприкінці оцінки відбувається формування нових колонок з результатами у вхідному датасеті для яких обчислюється похибка у кілометрах. Оброблений датасет зберігається у відповідну папку результатів. Статистика за похибкою відстані обраховується та записується у результуючі файли зображень з графіками та чисельними значеннями метрик у відповідних папках.
Очікуваний результат	Оцінка запускається, проходить, та завершується успішно, Розробник отримує файл обробленого датасету та файли статистики результатів оцінки моделі
Фактичний результат	Оцінка запускається, проходить, та завершується успішно, Розробник отримує файл обробленого датасету та файли статистики результатів оцінки моделі

Таблиця 3.4 – Тест 1.3

Тест	Прогнозування локації за текстом
Модуль	ПЗ Розробника
Номер тесту	1.3
Початковий стан системи	Розробник встановив ПЗ за інструкцією та відкрив консоль у кореневій папці проекту, у папці моделей знаходиться файл навченої моделі
Вхідні данні	Параметри моделі, назва файлу моделі, текст
Опис проведення тесту	Запускається прогнозування локації за текстом на базі готової моделі, параметризація відбувається за допомогою прапорів з аргументами, або за допомогою редагування змінних у скрипті вхідної точки прогнозування локації. На початку відбувається ініціація обгортки моделі з заданими параметрами, завантаження навченої моделі з файлу у пам’ять. Далі Розробнику пропонується ввести текст або продовжити прогнозування на базі стандартного запиту. Відбувається фільтрація та токенізація тексту, оцінюється правильність введеного тексту (довжина токенізованої послідовності не менше 1 і не перевищує 512 токенів) перед його завантаженням у модель. Пакет вхідних даних що складається з одного екземпляру подається на вхід, на виході отримані числові змінні проходять пост обробку для формування параметрів GMM що записуються у новий масив даних. В кінці у консоль виводиться чисельно-текстове представлення результату прогнозу, на базі якого генерується зображення розподілу місцезнаходження.

Перенесення Таблиці 3.4

Очікуваний результат	Прогноз місцезнаходження запускається, проходить, та завершується успішно, Розробник отримує файл зображення результату прогнозу
Фактичний результат	Прогноз місцезнаходження запускається, проходить, та завершується успішно, Розробник отримує файл зображення результату прогнозу

Таблиця 3.5 – Тест 2.1

Тест	Прогнозування локації за текстом
Модуль	ПЗ Користувача у вигляді Телеграм боту
Номер тесту	2.1
Початковий стан системи	Користувач запустив Телеграм бот, серверна частина застосунку працює, модель завантажена та ініційована на сервері
Вхідні данні	Текст

Перенесення Таблиці 3.5

Опис проведення тесту	Запускається прогнозування локації за текстом на базі готової моделі що завантажена до репозиторію на HuggingFace. Користувач вводить текст як аргумент команди «/predict», відбувається фільтрація та токенізація тексту, оцінюється правильність введеного тексту (довжина токенізованої послідовності не менше 1 і не перевищує 512 токенів) перед його завантаженням у модель. Пакет вхідних даних що складається з одного екземпляру подається на вхід, на виході отримані числові змінні проходять пост обробку для формування параметрів GMM що записуються у новий масив даних. В кінці у БД логів записується чисельно-текстове представлення результату прогнозу, вхідний текст та контакти Користувача. На базі отриманого прогнозу генерується зображення розподілу місцезнаходження на мапі Меркатора та надсилається Користувачеві разом з підписом чисельно-текстового представлення результату та вхідного тексту.
Очікуваний результат	Прогноз місцезнаходження запускається, проходить, та завершується успішно, Користувач отримує зображення результату прогнозу
Фактичний результат	Прогноз місцезнаходження запускається, проходить, та завершується успішно, Користувач отримує зображення результату прогнозу

3.3 Опис контрольного прикладу

В якості контрольного прикладу буде описано Тест 1.3 Прогнозування місцезнаходження за текстом в рамках використання ПЗ Розробником. Приклад прогнозування місцезнаходження буде показано на базі платформи Google Colab що є одним з видів консольного інтерфейсу Розробника.

Спочатку код гілки predict репозиторію GitHub клонується для локального використання, а всі залежності бібліотек Python завантажуються у віртуальне середовище як показано на Рисунку 3.3. Після встановлення залежностей віртуальне середовище Google Colab перезапускається.

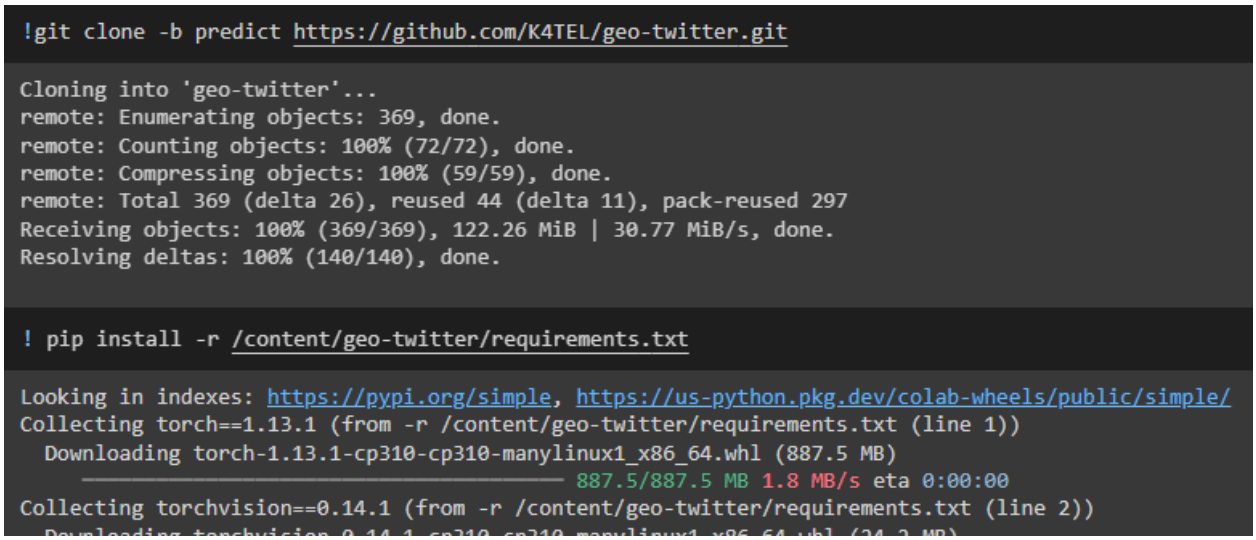


Рисунок 3.3 – Клонування проекту та встановлення залежностей

Далі папка склонованого проекту додається до системного шляху Google Colab, відбувається імпортування класів та функцій з основного файлу скрипту, встановлюються статичні змінні які потрібні для подальшого завантаження моделі як вказано на Рисунку 3.4. Використання моделі можливе за допомогою Pipeline трансформерів для попередньої обробки тексту та пост обробки результатів моделі, або за допомогою еквівалентних операцій без використання наслідника класу Pipeline.

Використовуючи функцію для завантаження моделі та шляхи до моделей на HuggingFace відбувається завантаження базової, а потім натренованої моделі для подальшого використання.

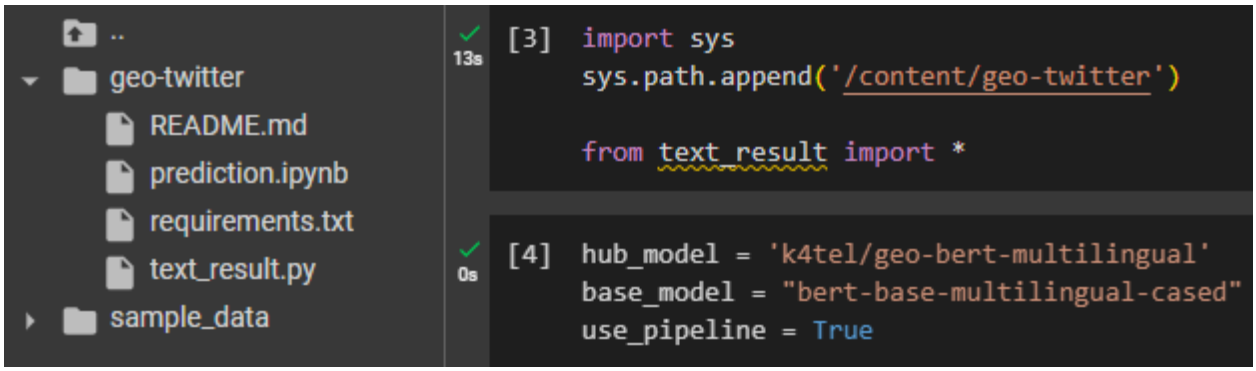


Рисунок 3.4 – Імпортування коду та встановлення статичних змінних для завантаження моделі

В результаті обгортка моделі налаштована на прогноз формату GMM з 5 піків надає доступ до готової моделі як показано на Рисунку 3.5. Різниця між використанням моделі за допомогою Pipeline або без нього не впливає на результат прогнозу – тільки на методи обробки вхідних та вихідних змінних моделі. Зокрема при його використанні обробка результатів відбувається за допомогою бібліотеки NumPy, в іншому випадку – за допомогою бібліотеки PyTorch.

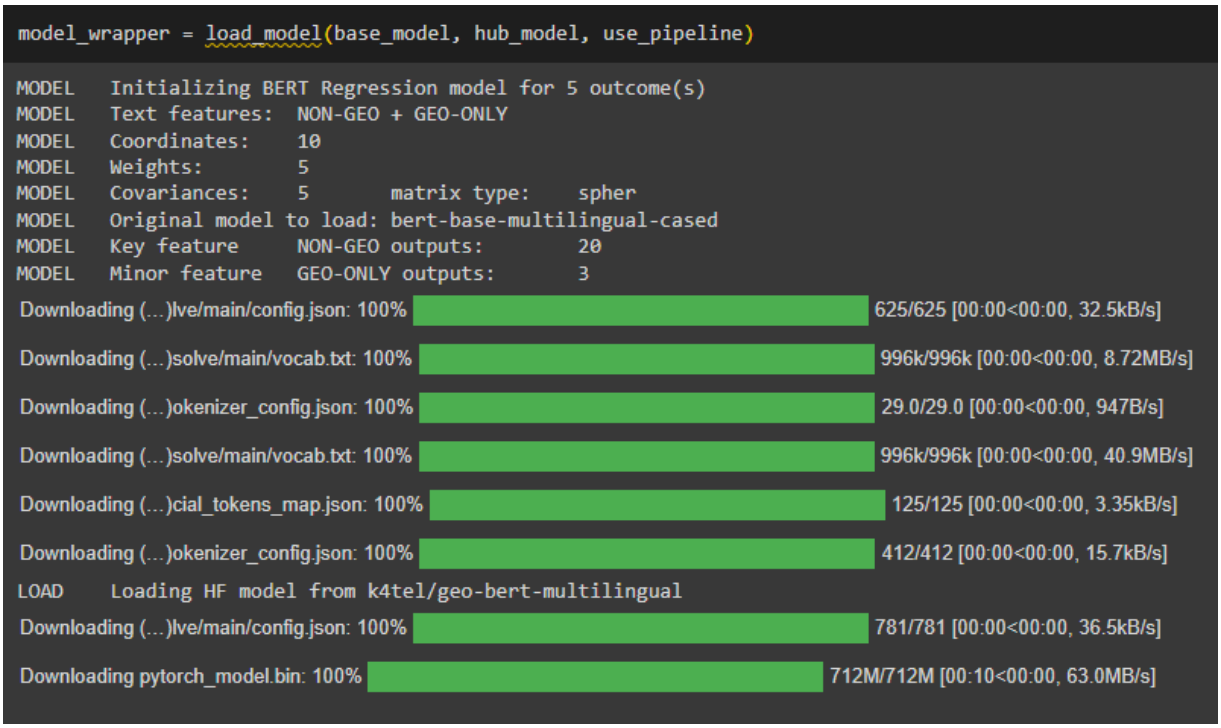


Рисунок 3.5 – Завантаження моделі з HuggingFace репозиторію

Далі встановлюється значення вхідної змінної тексту та опції фільтрації тексту перед завантаженням у модель. Фільтрація тексту відбувається для

видалення знаків пунктуації та посилань. Використовуючи стандартну функцію прогнозування місцезнаходження за текстом отримується результат прогнозу у чисельному вигляді параметрів GMM як показано на Рисунку 3.6. Після отримання результату, піки GMM сортуються за вагою, ті які не мають достатньої ваги видаляються, а ті що залишились виводяться у консоль без урахування їх параметру коваріації. Центри піків зі значною вагою представляють собою пари координат довготи та широти, а їх вага переводиться у відсотки.

```
text = "Героям слава, слава нації, смерть ворогам"
filter = True

result = text_prediction(model_wrapper, text, use_pipeline, filter)

ind = np.argwhere(np.round(result.weights[0, :] * 100, 2) > 0)
significant = result.means[0, ind].reshape(-1, 2)
weights = result.weights[0, ind].flatten()

sig_weights = weights[weights > 0]

print(f"RESULT\t{len(sig_weights)} significant prediction outcome(s):")
for i in range(len(sig_weights)):
    point = f"lon: {'  lat: '.join(map(str, significant[i]))}"
    weight = str(np.round(sig_weights[i] * 100, 2))
    print(f"\tOut {i + 1}\t{weight}%\t-\t{point}")
```

```
TEXT    Filtering text: Героям слава, слава нації, смерть ворогам
RESULT  Post-processing raw model outputs: tensor([[  9.5680,  25.0352,  14.6391,  42.3848,  29.3000,  47.8244,  24.0846,
          -4.2688,  31.5437,  26.4489, -18.9937,  -1.6548,   5.8224, -19.1575,
         -18.4819,   3.5919, 136.6868,   3.0500,  -6.4221,  58.3281]])
RESULT  Sorting all outputs for 5 outcomes by probabilistic weights
RESULT  2 significant prediction outcome(s):
        Out 1   99.94%  -       lon: 29.300025939941406  lat: 47.82440948486328
        Out 2   0.06%   -       lon: 14.6390962600708  lat: 42.384788513183594
```

Рисунок 3.6 – Прогнозування місцезнаходження за текстом

Фінальним кроком отримання прогнозу місцезнаходження за текстом є генерація зображення двовимірного розподілу GMM на мапі Меркатора як показано на Рисунку 3.7. При цьому на мапі та в легенді буде позначено лише значні за вагою піки GMM, а вхідний текст буде відображено в назві графіку.

```
visual = ResultVisuals(result)
visual.text_map_result()
```

Рисунок 3.7 – Виклик візуалізації результату прогнозу

Результатом візуалізації прогнозу місцезнаходження є спектри значень функцій PDF та їх логарифмів для отриманого GMM що задаються

кольоровим градієнтом на мапі як показано на Рисунку 3.8. Значення логарифму вірогідності отриманого розподілу наноситься на повномасштабну мапу Меркатора, а значення PDF наноситься на приближений до най вірогіднішого піку регіон мапи на якій відображено адміністративні та географічні кордони країн та водойм.

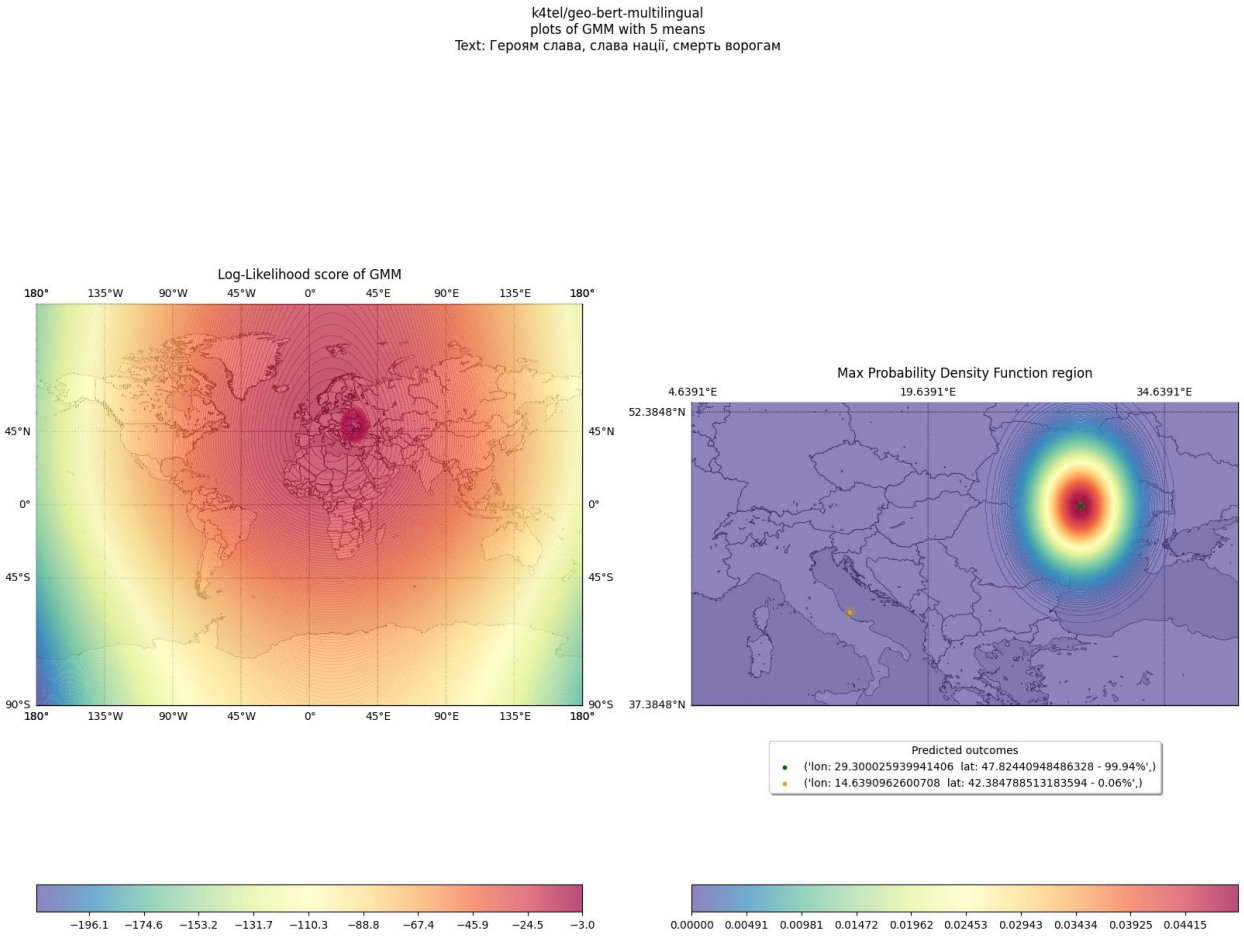


Рисунок 3.8 – Візуалізації результату прогнозу місцезнаходження

Наданий контрольний приклад прогнозування місцезнаходження за текстом також відображає базову логіку роботи Телеграм боту як користувацького інтерфейсу для демонстрації найкращої з навчених моделей.

Висновки до розділу

Було описано процедури оцінки якості ПЗ Розробника, зокрема метрики точності алгоритмів машинного навчання для прогнозування місцезнаходження. Запропоновану модель було порівняно з попередніми роботами в рамках вирішення задач прогнозування домашнього місцезнаходження користувача та місцезнаходження твіта.

Було надано варіанти базових тестів ПЗ Розробника та ПЗ Користувача у вигляді Телеграм боту для демонстрації роботи моделі. Більш того, було детально описано контрольний приклад прогнозування місцезнаходження за текстом з використанням консольного інтерфейсу та готової моделі завантаженої у репозиторій HuggingFace.

Підсумовуючи, було виділено дві геопросторові метрики оцінки моделей (SAE та Acc@161) та проаналізовано статистику результатів найкращої моделі.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розгортання ПЗ потребує лише користувацький інтерфейс Телеграм боту, клієнтська частина якого працює дистанційно на сервері Телеграм, в той час як його серверну частину було вирішено розгорнути на сервері на базі ОС Linux. Консольний інтерфейс розробника не потребує окремого розгортання з огляду на відсутність клієнт-серверної складової у цьому компоненті ПЗ.

Розгортання починається коли всі файли репозиторію GitHub гілки tgbot перенесено на сервер за допомогою ssh підключення та захищених протоколів обміну пакетами даних (scp) де встановлено базовий інтерпретер мови Python. Тоді на сервері створюється Docker image за допомогою інструкцій наданих у Dockerfile та запуску інструкцій у buildDocker. Цей image розгортається за допомогою інструкцій у runDocker що автоматично запускає tmux середовище для роботи Телеграм боту.

Під час розгортання серверної частини застосунку інструкції створення Docker image автоматично встановлюють міінімальний набір потрібних pip бібліотек Python. А при розгортанні image відбувається запуск прослуховувача подій на базі Telegram Bot API та підвантаження моделі з репозиторію HuggingFace. Вигляд консольного виводу tmux середовища після вдалого запуску Телеграм боту показано на Рисунку 4.1

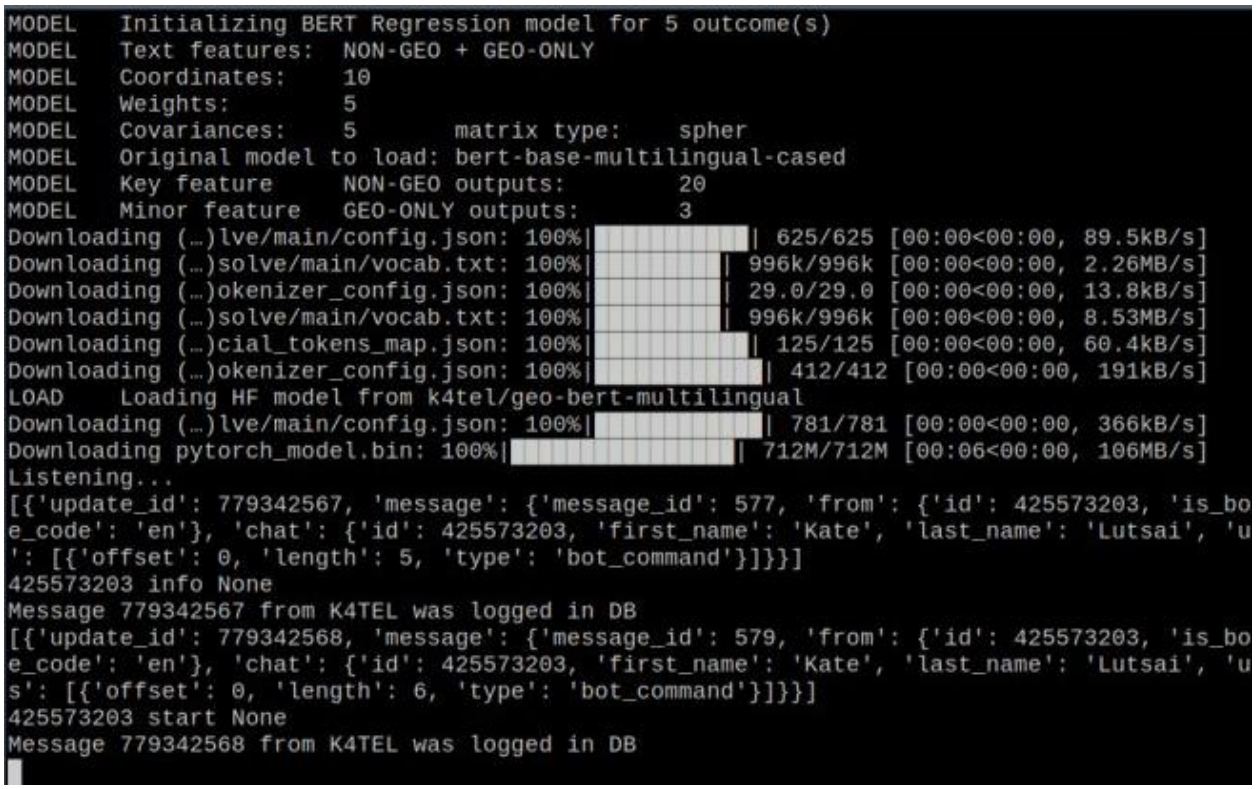


Рисунок 4.1 – Інформація про розгортання серверу Телеграм боту

4.2 Підтримка програмного забезпечення

Розробники повинні мати можливість доступу до частини ПЗ що включає алгоритми машинного навчання моделей для незалежного запуску на власних локальних машинах. Підтримка цього компоненту ПЗ передбачає оновлення гілки main репозиторію GitHub для підлаштування до майбутніх оновлень бібліотек, супровід коду інструкціями до самостійного запуску, та надання персональної підтримки через електронну пошту.

Також розробники повинні мати можливість доступу до готової моделі завантаженої на HuggingFace та її консольного інтерфейсу для самостійного запуску прогнозування місцезнаходження за текстом. Для підтримки цього компоненту ПЗ узгоджене з основною гілкою оновлення гілки predict репозиторію GitHub буде достатньо.

Звичайні користувачі повинні мати можливість доступу до Телеграм боту для отримання прогнозу місцезнаходження за текстом. Для цього роботу серверної частини пропонується підтримувати за допомогою оновлення

локального набору файлів гілки tgbot репозиторію GitHub та за потреби рестарту боту.

Висновки до розділу

Оскільки ПЗ складається з двох інтерфейсів – консоль розробника та Телеграм бот для звичайних користувачів – але сервер-клієнт складова присутня лише в користувацькому інтерфейсі, то процес розгортання було описано лише для нього. Підтримку усього ПЗ реалізовано за допомогою оновлення репозиторію GitHub для підлаштування під нові версії pip бібліотек Python, а підтримка Телеграм боту відбувається на локальному приватному сервері. При подальшому оновленні даного ПЗ внесення значних правок у процес навчання або обробки результатів моделі не передбачено, тому оновлення репозиторію HuggingFace не було описано.

ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано ПЗ для Розробника у вигляді консольного інтерфейсу та ПЗ для звичайного Користувача у вигляді Телеграм боту для демонстрації роботи моделі. Поставлені задачі проєктування ПЗ для навчання, оцінки та використання моделей з метою покращення точності прогнозу місцезнаходження за текстом було виконано у повному об’ємі.

Зокрема, ця робота була спрямована на вивчення методів машинного навчання для розв'язання задачі прогнозування місцезнаходження за коротким текстом методами NLP із застосуванням нейронних мереж на основі BERT. Реалізована модифікація архітектури моделі виграє завдяки простоті налаштування, необхідного для навчання власних моделей під конкретну задачу Розробника. Запропоновані моделі є гнучкими до будь-якого рівня геопросторової деталізації (світ, країна, місто і т.д.) шляхом зміни набору даних для навчання та мовної специфіки базової попередньо навченої BERT-моделі.

На основі оцінки якості прогнозів моделі за допомогою загальноприйнятих метрик (SAE та Acc@161), можна зробити висновок що модель відповідає сучасному рівню наукових і технічних знань. Для всесвітнього рівня географічної деталізації було проведено оцінку моделей та отримано медіанне значення відстані похибки менше 30 кілометрів, при цьому відсоток прогнозів похибка яких не перевищує 161 кілометру становив майже 80%.

В якості середовища розробки обрано мову Python та ОС на базі Linux, де встановлюються всі потрібні бібліотеки для роботи ПЗ.

У якості БД серверної частини Телеграм боту використано SQLite для логу запитів користувачів та відповідей моделі. Для ПЗ Розробника було використано стандартизований формат збереження датасетів у єдиному файлі табличного вигляду без застосування окремих СУБД.

Після реалізації застосунку він був протестований на пристроях з різними версіями Linux та Python щоб переконатися, що ПЗ Розробника правильно працює на різних пристроях. Функціонал ПЗ Розробника та Користувача було мануально протестовано під час роботи з консольним та Телеграм бот інтерфейсами.

Вперше реалізовано можливість багатозадачного навчання для паралельного навчання на другорядній змінній яка містить автоматично згенерований опис локації твіта, що дозволило збільшити точність прогнозів моделей.

Однією з проблем, згаданих раніше, була альтернатива довідникам – словникам географічних індексів, які використовувалися в попередніх роботах і містили точні терміни (географічні назви) для географічних об'єктів та їхні правильні координати. Враховуючи зашумленість даних, що генеруються користувачем у типових вхідних даних моделі, ідея подачі в модель узгодженої бази знань про географічні об'єкти виявилося практичною і було успішно реалізовано. Найефективніші моделі були навчені багатозадачним методом що надає можливість паралельного навчання із загальними втратами на пакет. Найкращою стратегією було розділення на ключову NON-GEO (завжди присутні "текст" і "користувач") і другорядну GEO-ONLY ("місце" присутнє тільки в твітах з геотегами) текстові змінні. На практиці, кожна вхідна змінна мала свій окремий шар лінійної регресії, що обгортав базову модель, які відрізнялися за кількістю результатів прогнозування, але збігалися за типом функції втрат для кожної змінної (геопросторова/імовірнісна). Оцінювання проводилося з використанням лише шару обгортки ключової змінної без урахування шару обгортки другорядних змінних, який використовувався лише під час навчання.

Такий підхід дозволив зберегти формат загальних вхідних даних моделі та зв’язати автоматично згенеровані описи місць з локаціями та текстом заданими користувачами Твітера. Пряма конкатенація текстового вмісту та метаданих "місця" вимагала б небажаної рандомізації всіх слів у рядку перед

токенізацією, так щоб моделі не навчилися звертати увагу лише на необов'язкову частину (метадані "місця") вхідної послідовності. Модель навчена на об'єднанні "text", "user" і "place", як на ключовій змінній, показала гірші результати, ніж модель, навчена на даних, розділених на ключові та другорядні змінні. Більше того, модель, навчена виключно на ключовій NON-GEO змінній без урахування контексту місця, продемонструвала ще гірші просторові помилки під час оцінювання. Аналогічно, набір даних GeoText не має доступних метаданих "місцевості", що призвело до вищих просторових помилок порівняно з іншими наборами даних, відтвореними на місцевості, які містили всі необхідні дані, як показано в Таблиці 3.1. Таким чином, запропонована методологія багатозадачного навчання продемонструвала значне покращення точності прогнозів і рекомендована до використання, якщо це можливо.

Вперше використано оцінку набору прогнозів твітів користувача у форматі GMM, що дозволило обійти ліміт 512 токенів вхідної послідовності при прогнозуванні домашнього місцезнаходження користувача за наведеними у Додатку Д формулами. Хоча прогнозування домашнього місцезнаходження користувача не було основним показником ефективності, результати, наведені в Таблиці 3.1, показують вищу просторову точність ніж у прогнозування місцезнаходження твіту. Таким чином, моделі типу PMOP пропонуються для прогнозування домашнього місцезнаходження користувача як рішення на основі BERT, яке перевершує більшість попередніх робіт за точністю прогнозування.

Модифіковано процедуру обчислення імовірнісних втрат за допомогою рівняння (LBSP) обмеження параметру коваріації GMM, що дозволило компонувати значення геопросторових та імовірнісних втрат навчальних пакетів для покращення точності прогнозів моделей. Оскільки імовірнісна функція втрат обчислювала від'ємну логарифмічну ймовірність (NLLH) того, що справжня точка місцезнаходження вписується в передбачений розподіл, вона залежала від щільності ймовірності (PDF). У випадку наближення $\sigma_{\hat{c}}$ до

свого мінімуму в 0, PDF перевищувало 1, що призводило до від'ємних значень функції втрат, як показано на Рисунку 2.5. З точки зору імовірнісних моделей, сумарні втрати залежали як від просторової, так і від імовірнісної складової, тому остання набирала більшої ваги (інерції) під час навчання. Як наслідок, модель приділяла менше уваги квадрату відстані евклідової похибки, що описується рівнянням (SED), що негативно впливало на геопросторову точність прогнозів. Було запропоновано рішення обмежили PDF інтервалом [0, 1], встановивши мінімум $\sigma_{\hat{c}}$ рівним $\frac{1}{2\pi}$ так, що ймовірнісні втрати залишаються в межах додатніх чисел. Оскільки коваріаційний параметр описує невизначеність моделі щодо прогнозованої точки місцезнаходження, в найкращому випадку ймовірнісні втрати залежатимуть головним чином від просторової помилки, як описано в Підрозділі 2.3.

Набуло подальший розвиток використання нейронних мереж для прогнозування місцезнаходження у форматі параметрів GMM та використання моделей регресії на базі BERT, що дозволило покращити точність результатів.

Що стосується моніторингу місцезнаходження користувачів у реальному часі, то Юань у роботі [42] зосередився на відстеженні переміщення окремих осіб або груп у часі. Моніторинг геолокації користувачів у часі може надати цінну інформацію для різноманітних застосувань, таких як відстеження спалахів захворювань, аналіз моделей міської мобільності та надання послуг на основі визначення місцезнаходження. Така аналітична база може бути побудована на основі запропонованих моделей, але в цьому дослідженні ми розглянули лише загальновідому задачу прогнозування домашнього місцезнаходження користувача без подальшого аналізу динаміки результатів.

Крім того, існує кілька різних моделей світу, які можна використовувати для прогнозування. Хоча проекція Меркатора зазвичай використовується завдяки геолокації Twitter, що зберігається у форматі WGS84, інші проекції, такі як проекція Робінсона, конічна проекція та проекція Вінкеля-Тріпеля, слід розглядати як можливі альтернативи в майбутніх дослідженнях.

Аналогічно, запропоновані методи машинного навчання можуть бути використані для будь-якої базової моделі, окрім згаданих варіацій BERT. Це потребувало б адаптації реалізації шару обгортки відповідно до форми вихідного вектора пулера обраної моделі, оскільки поточний підхід налаштований на логіку лінійної регресії, що перетворює вектор класифікації розміру 768, спільний для всіх моделей на основі BERT, у заздалегідь визначену кількість неперервних числових значень.

Загалом, аналіз настроїв на основі геолокації є важливим інструментом для розуміння громадської думки, соціальної динаміки та закономірностей, допомагає дослідникам приймати рішення на основі даних, а також підтримує роботу різних секторів – від маркетингу до державного управління. У цьому дослідженні представлено підхід на основі NLP для оцінки геолокації шляхом обробки коротких текстових корпусів, таких як пости в соціальних мережах, наприклад, у Twitter. Запропоноване рішення використовує багатозадачне навчання (ключові та другорядні вхідні змінні), контекстні дані (метадані про користувача та місце) та імовірнісний вивід (двовимірний розподіл GMM) для досягнення вищої просторової точності в задачах прогнозування місцезнаходження твіту та домашнього місцезнаходження користувача. Таким чином, ми робимо свій внесок у сферу аналізу великих даних з гнучким налаштуванням географічної деталізації моделей регресії на основі BERT.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) ARAFAT, T. A., BUDI, I., MAHENDRA, R., AND SALEHAH, D. A. Demographic analysis of candidates supporter in twitter during indonesian presidential election 2019. In 2020 International Conference on ICT for Smart Society (ICISS) (2020), IEEE, pp. 1–6.

2) BAKERMAN, J., PAZDERNIK, K., WILSON, A., FAIRCHILD, G., AND BAHRAN, R. Twitter geolocation: A hybrid approach. ACM Transactions on Knowledge Discovery from Data (TKDD) 12, 3 (2018), 1–17.

3) CHA, M., GWON, Y., AND KUNG, H. Twitter geolocation and regional classification via sparse coding. In Proceedings of the International AAAI Conference on Web and Social Media (2015), vol. 9, pp. 582–585.

4) CHENG, Z., CAVERLEE, J., AND LEE, K. You are where you tweet: a content-based approach to geo-locating twitter users. In Proceedings of the 19th ACM international conference on Information and knowledge management (2010), pp. 759–768.

5) COMPTON, R., JURGENS, D., AND ALLEN, D. Geotagging one hundred million twitter accounts with total variation minimization. In 2014 IEEE international conference on Big data (big data) (2014), IEEE, pp. 393–401.

6) DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).

7) DO, T. H., NGUYEN, D. M., TSILIGIANNI, E., CORNELIS, B., AND DELIGIANNIS, N. Twitter user geolocation using deep multiview learning. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2018), IEEE, pp. 6304– 6308.

8) EBRAHIMI, M., SHAFIEIBAVANI, E., WONG, R., AND CHEN, F. Twitter user geolocation by filtering of highly mentioned users. Journal of the Association for Information Science and Technology 69, 7 (2018), 879–889.

9) EISENSTEIN, J., AHMED, A., AND XING, E. P. Sparse additive generative models of text. In Proceedings of the 28th international conference on machine learning (ICML-11) (2011), pp. 1041–1048.

10) EISENSTEIN, J., O'CONNOR, B., SMITH, N. A., AND XING, E. A latent variable model for geographic lexical variation. In Proceedings of the 2010 conference on empirical methods in natural language processing (2010), pp. 1277–1287.

11) FLATOW, D., NAAMAN, M., XIE, K. E., VOLKOVICH, Y., AND KANZA, Y. On the accuracy of hyper-local geotagging of social media content. In Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (2015), pp. 127–136.

12) HAN, B., COOK, P., AND BALDWIN, T. Geolocation prediction in social media data by finding location indicative words. In Proceedings of COLING 2012 (2012), pp. 1045– 1062.

13) HAN, B., COOK, P., AND BALDWIN, T. Text-based twitter user geolocation prediction. Journal of Artificial Intelligence Research 49 (2014), 451–500.

14) HUANG, B., AND CARLEY, K. M. A hierarchical location prediction neural network for twitter user geolocation. arXiv preprint arXiv:1910.12941 (2019).

15) HULDEN, M., SILFVERBERG, M., AND FRANCOM, J. Kernel density estimation for text-based geolocation. In Proceedings of the AAAI conference on artificial intelligence (2015), vol. 29.

16) ISO, H., WAKAMIYA, S., AND ARAMAKI, E. Density estimation for geolocation via convolutional mixture density network. arXiv preprint arXiv:1705.02750 (2017).

17) JURGENS, D. That's what friends are for: Inferring location in online social media plat- forms based on social relationships. In Proceedings of the International AAAI Conference on Web and Social Media (2013), vol. 7, pp. 273–282.

18) KINSELLA, S., MURDOCK, V., AND O'HARE, N. " i'm eating a sandwich in glasgow" modeling locations with tweets. In Proceedings of the 3rd international workshop on Search and mining user-generated contents (2011), pp. 61–68.

19) KONG, L., LIU, Z., AND HUANG, Y. Spot: Locating social media users based on social network context. Proceedings of the VLDB Endowment 7, 13 (2014), 1681–1684.

20) LI, M., LIM, K. H., GUO, T., AND LIU, J. A transformer-based framework for poi- level social post geolocation. arXiv preprint arXiv:2211.01336 (2022).

21) LIU, J., AND INKPEN, D. Estimating user location in social media with stacked de- noising auto-encoders. In Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing (2015), pp. 201–210.

22) MCGEE, J., CAVERLEE, J., AND CHENG, Z. Location prediction in social media based on tie strength. In Proceedings of the 22nd ACM international conference on Information & Knowledge Management (2013), pp. 459–468.

23) MELO, F., AND MARTINS, B. Geocoding textual documents through the usage of hier- archical classifiers. In Proceedings of the 9th Workshop on Geographic Information Retrieval (2015), pp. 1–9.

24) MIURA, Y., TANIGUCHI, M., TANIGUCHI, T., AND OHKUMA, T. Unifying text, metadata, and user network representations with a neural network for geolocation pre- diction. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (2017), pp. 1260–1272.

25) MIYAZAKI, T., RAHIMI, A., COHN, T., AND BALDWIN, T. Twitter geolocation using knowledge-based methods. In Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text (2018), pp. 7–16.

26) PRIEDHORSKY, R., CULOTTA, A., AND DEL VALLE, S. Y. Inferring the origin locations of tweets with quantitative confidence. In Proceedings of the

17th ACM conference on Computer supported cooperative work & social computing (2014), pp. 1523–1536.

27) RAHIMI, A., BALDWIN, T., AND COHN, T. Continuous representation of location for geolocation and lexical dialectology using mixture density networks. arXiv preprint arXiv:1708.04358 (2017).

28) RAHIMI, A., COHN, T., AND BALDWIN, T. A neural model for user geolocation and lexical dialectology. arXiv preprint arXiv:1704.04008 (2017).

29) RAHIMI, A., VU, D., COHN, T., AND BALDWIN, T. Exploiting text and network con- text for geolocation of social media users. arXiv preprint arXiv:1506.04803 (2015).

30) ROLLER, S., SPERIOSU, M., RALLAPALLI, S., WING, B., AND BALDRIDGE, J. Supervised text-based geolocation using language models on an adaptive grid. In Proceed- ings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning (2012), pp. 1500–1510.

31) RYOO, K., AND MOON, S. Inferring twitter user locations with 10 km accuracy. In Proceedings of the 23rd International Conference on World Wide Web (2014), pp. 643–648.

32) SADILEK, A., KAUTZ, H., AND BIGHAM, J. P. Finding your friends and following them to where you are. In Proceedings of the fifth ACM international conference on Web search and data mining (2012), pp. 723–732.

33) SCHERRER, Y., AND LJUBEŠIC´ , N. Social media variety geolocation with geobert. In Proceedings of the Eighth Workshop on NLP for Similar Languages, Varieties and Dialects (2021), The Association for Computational Linguistics.

34) SCHULZ, A., HADJAKOS, A., PAULHEIM, H., NACHTWEY, J., AND MÜHLHÄUSER, M. A multi-indicator approach for geolocalization of tweets. In Proceedings of the International AAAI Conference on Web and Social Media (2013), vol. 7, pp. 573–582.

35) SIMANJUNTAK, L. F., MAHENDRA, R., AND YULIANTI, E. We know you are living in bali: Location prediction of twitter users using bert language model. Big Data and Cognitive Computing 6, 3 (2022), 77.

36) VILLEGAS, D. S., PREOT¸ IUC-PIETRO, D., AND ALETRAS, N. Point-of-interest type inference from social media text. arXiv preprint arXiv:2009.14734 (2020).

37) WAKAMIYA, S., KAWAI, Y., ARAMAKI, E., ET AL. Twitter-based influenza detection after flu peak via tweets with indirect information: text mining study. JMIR public health and surveillance 4, 3 (2018), e8627.

38) WING, B., AND BALDRIDGE, J. Simple supervised document geolocation with geodesic grids. In Proceedings of the 49th annual meeting of the association for compu- tational linguistics: Human language technologies (2011), pp. 955–964.

39) WING, B., AND BALDRIDGE, J. Hierarchical discriminative classification for text- based geolocation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (2014), pp. 336–348.

40) YAMAGUCHI, Y., AMAGASA, T., AND KITAGAWA, H. Landmark-based user location inference in social media. In Proceedings of the first ACM conference on Online social networks (2013), pp. 223–234.

41) YAQUB, U., SHARMA, N., PABREJA, R., CHUN, S. A., ATLURI, V., AND VAIDYA, J. Location-based sentiment analyses and visualization of twitter election data. Digital Government: Research and Practice 1, 2 (2020), 1–19.

42) YUAN, Q., CONG, G., MA, Z., SUN, A., AND THALMANN, N. M. Who, where, when and what: discover spatio-temporal topics for twitter users. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (2013), pp. 605–613.

43) ZHENG, C., JIANG, J.-Y., ZHOU, Y., YOUNG, S. D., AND WANG, W. Social media user geolocation via hybrid attention. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (2020), pp. 1641–1644.

44) ZHENG, X., HAN, J., AND SUN, A. A survey of location prediction on twitter. IEEE Transactions on Knowledge and Data Engineering 30, 9 (2018), 1652–1671.

45) ZHONG, T., WANG, T., WANG, J., WU, J., AND ZHOU, F. Multiple-aspect attentional graph neural networks for online social network user localization. IEEE Access 8 (2020), 95223–95234.

46) ZHOU, F., WANG, T., ZHONG, T., AND TRAJCEVSKI, G. Identifying user geolocation with hierarchical graph neural networks and explainable fusion. Information Fusion 81 (2022), 1–13. https://doi.org/10.1016/j.inffus.2021.11.004.

ДОДАТОК А

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015398487

Дата перевірки:
02.06.2023 16:37:10 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2023 16:38:31 EEST

ID користувача:
76913

Назва документа: **ІТ-91_Луцай_ПЗ**

Кількість сторінок: **83** Кількість слів: **15550** Кількість символів: **116335** Розмір файлу: **3.60 MB** ID файлу: **1015062486**

4.08%
Схожість

Найбільша схожість: **1.43%** з джерелом з Бібліотеки (ID файлу: **1015058481**)

2% Джерела з Інтернету	130	Сторінка 85
3.86% Джерела з Бібліотеки	199	Сторінка 87

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

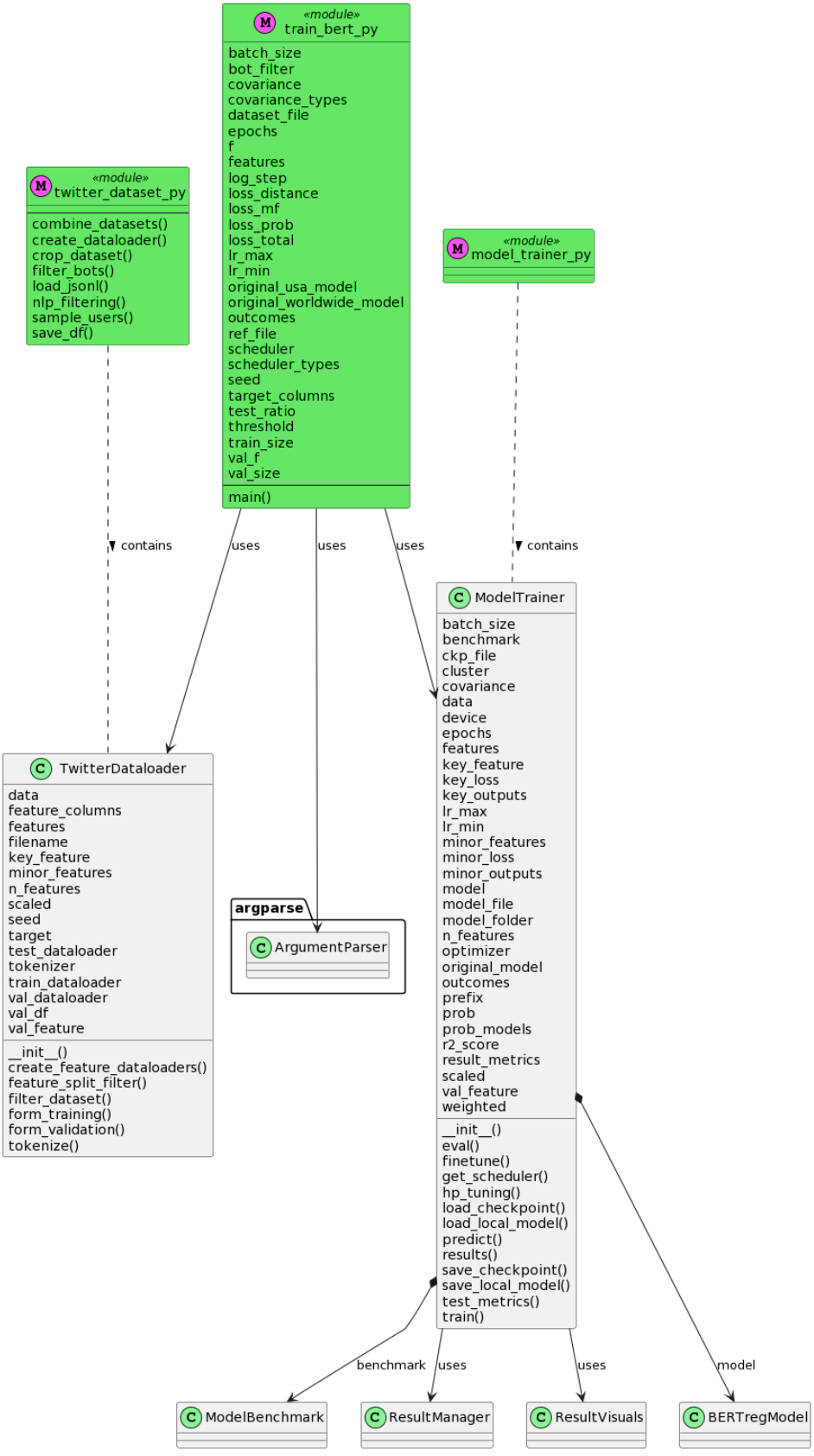
Модифікації

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

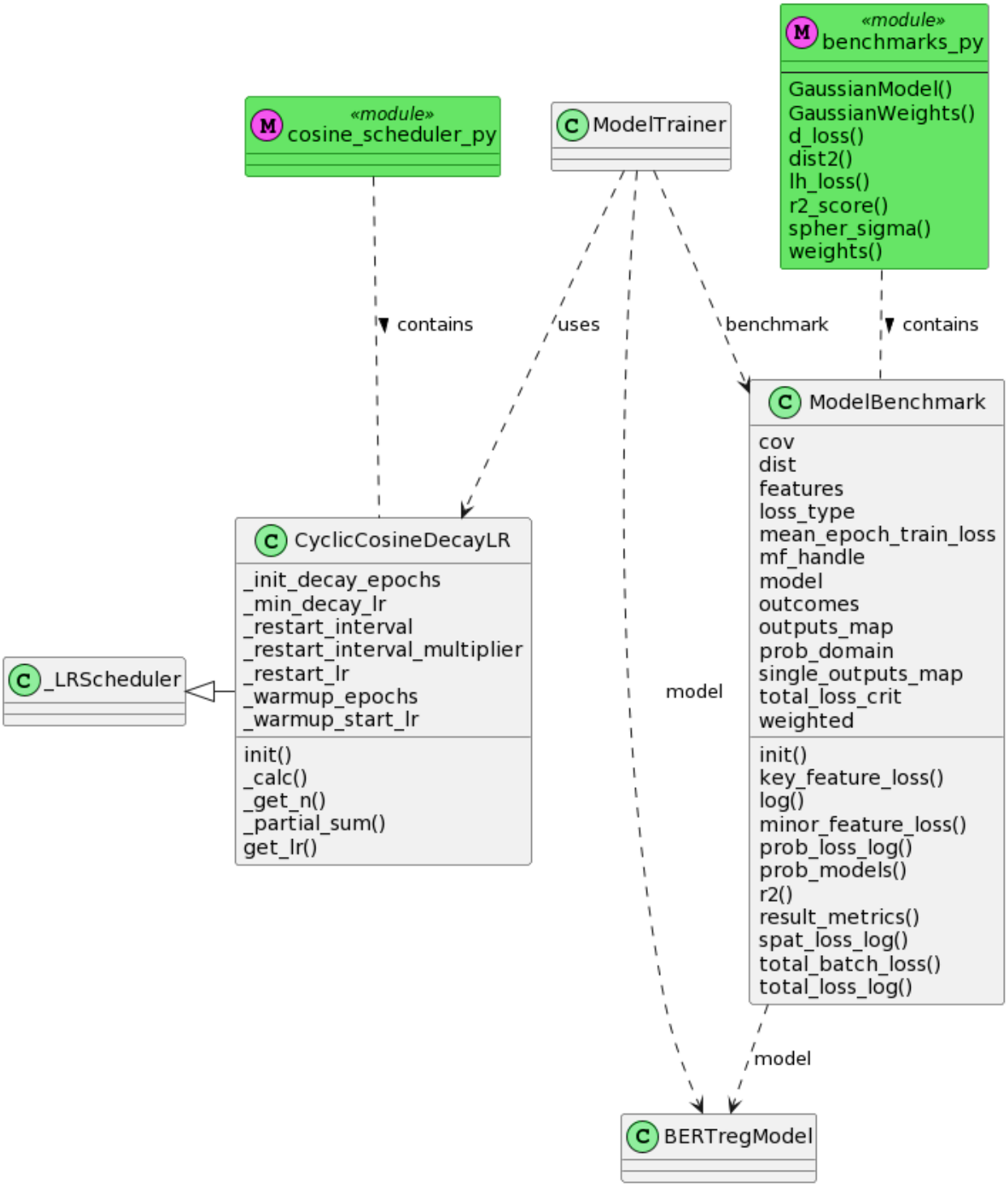
Замінені символи	2

ДОДАТОК Б

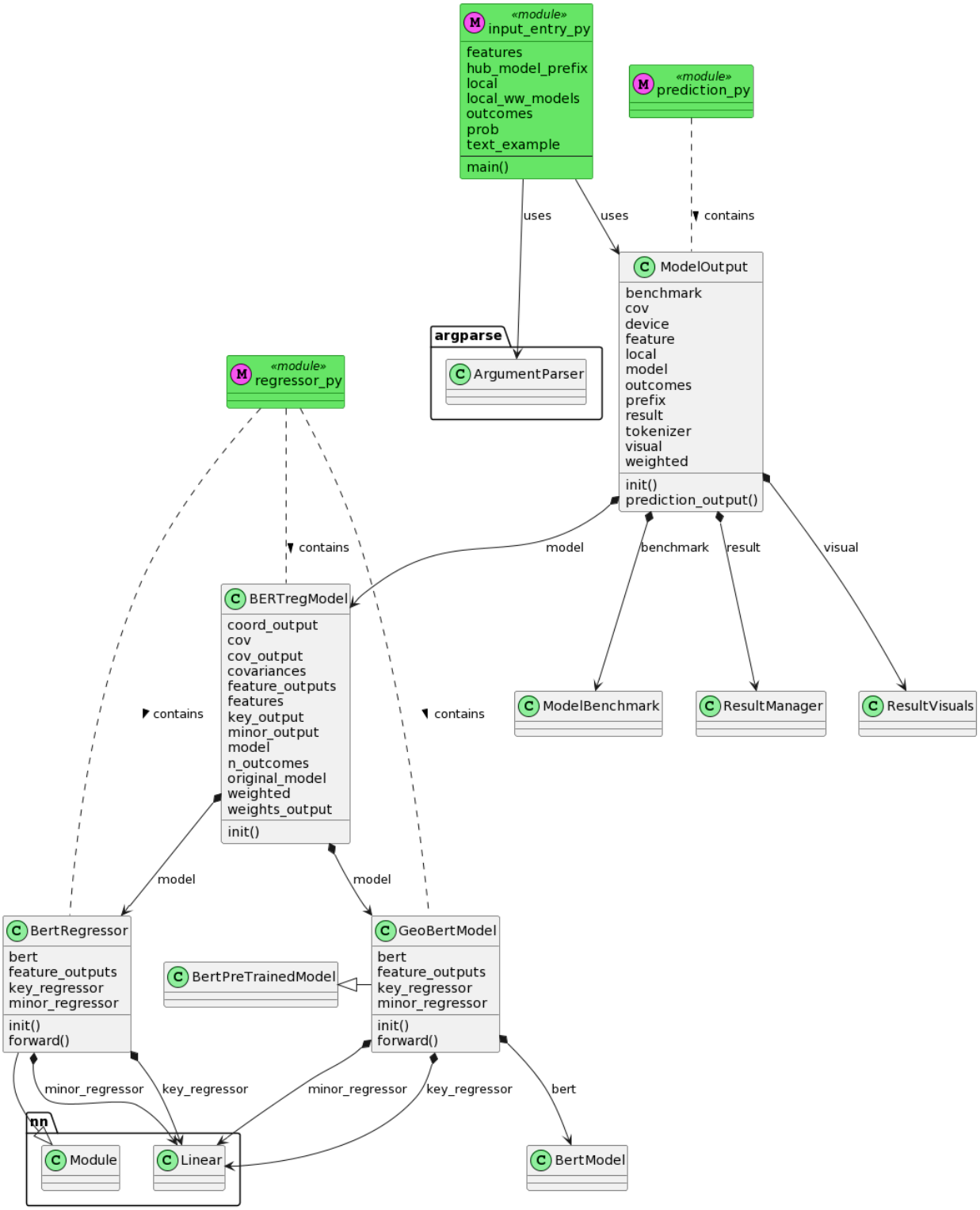
Структурні схеми класів



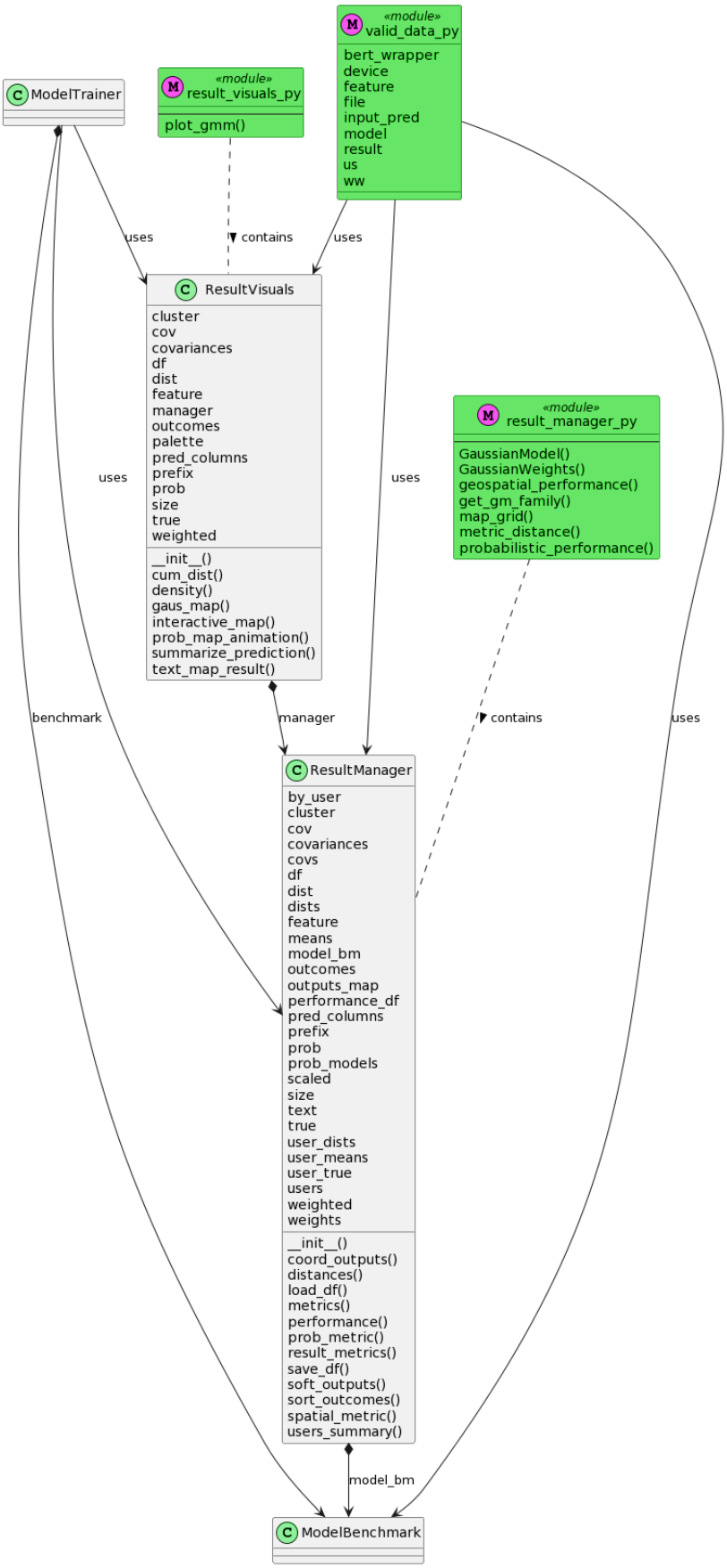
ModelTrainer та TwitterDataloader



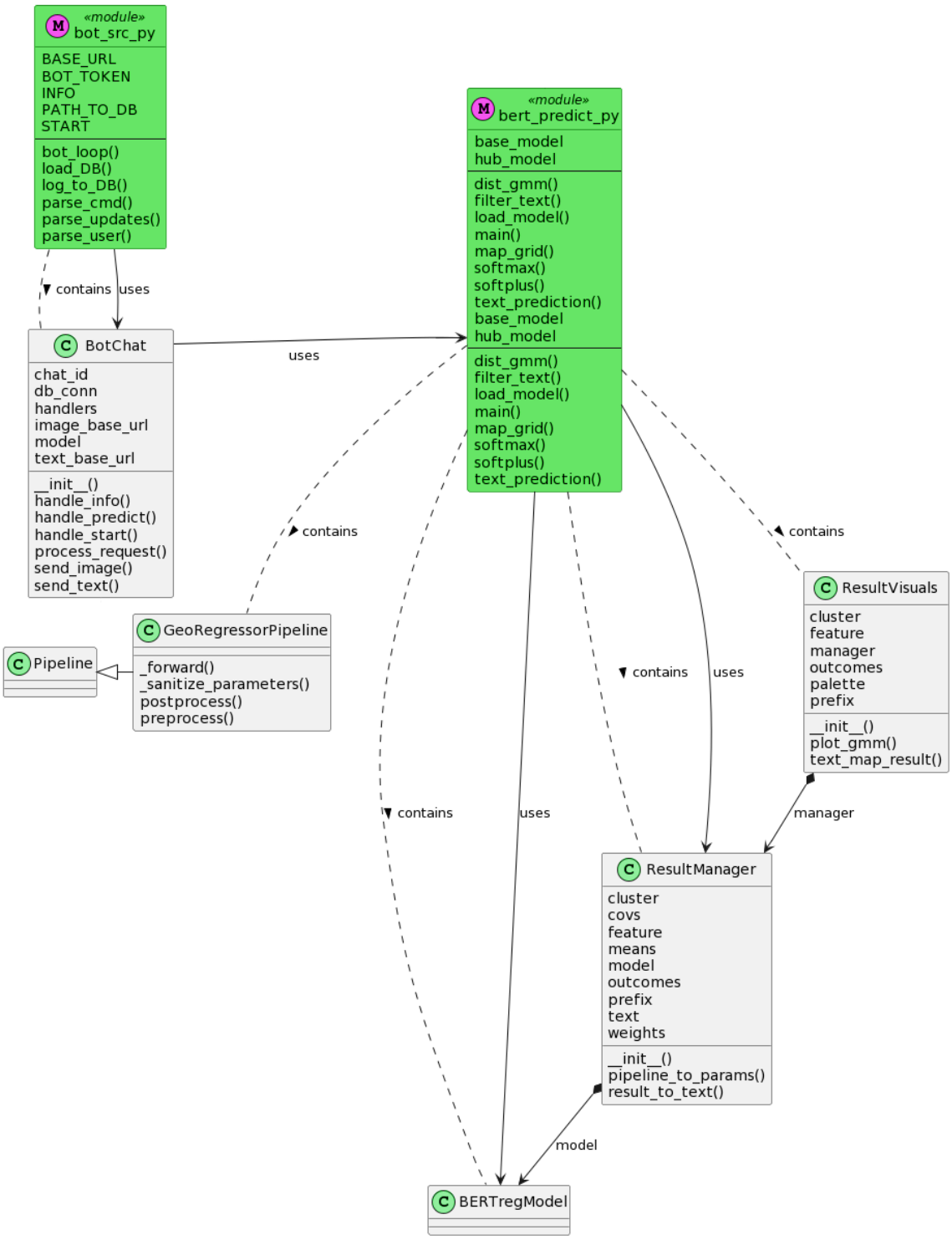
CyclicCosineDecayLR та ModelBenchmark



ModelOutput, BERTRegModel, BertRegressor та GeoBertModel



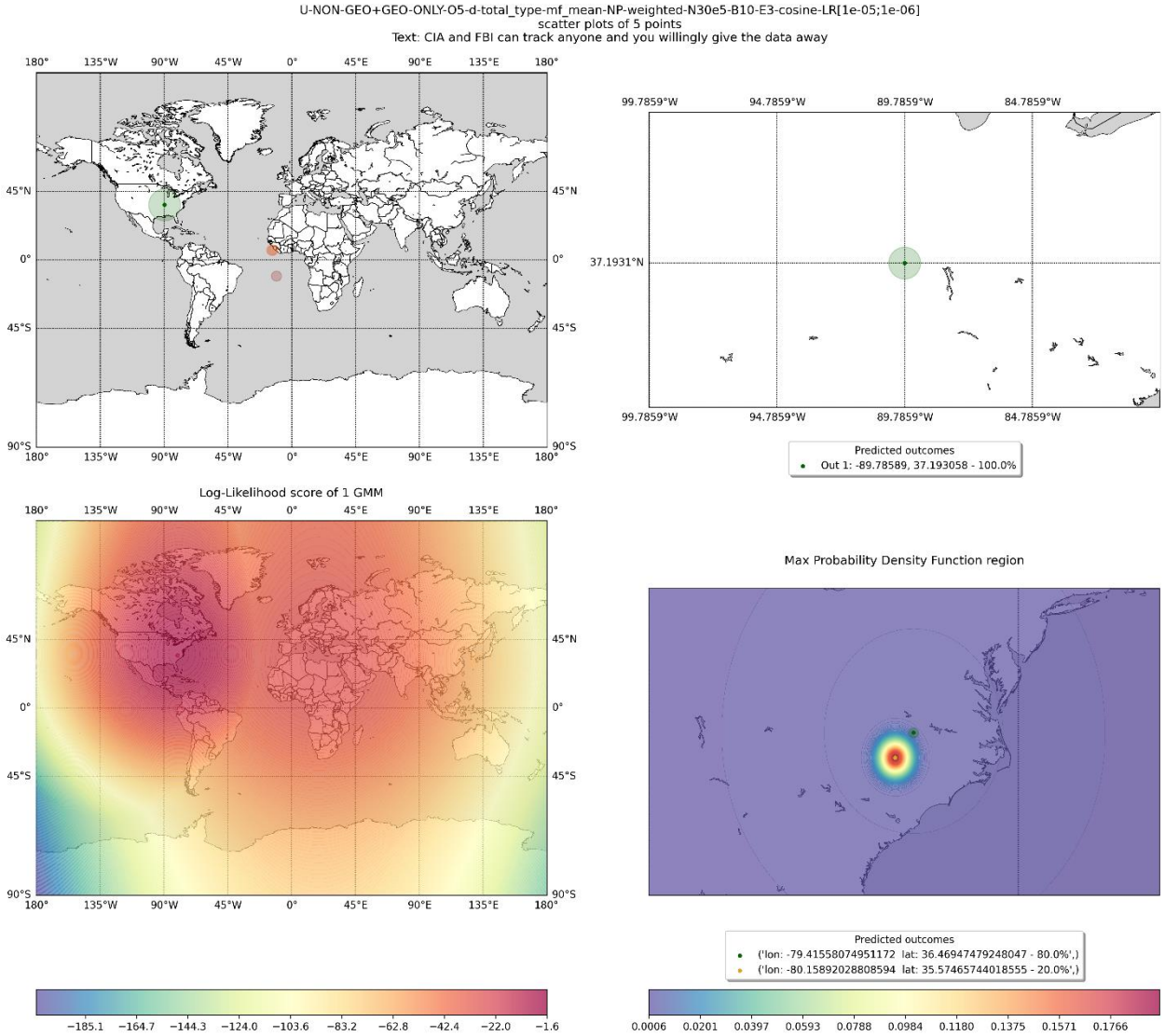
ResultManager та ResultVisuals



BotChat, GeoRegressorPipeline, та урізаних ResultVisuals, ResultManager

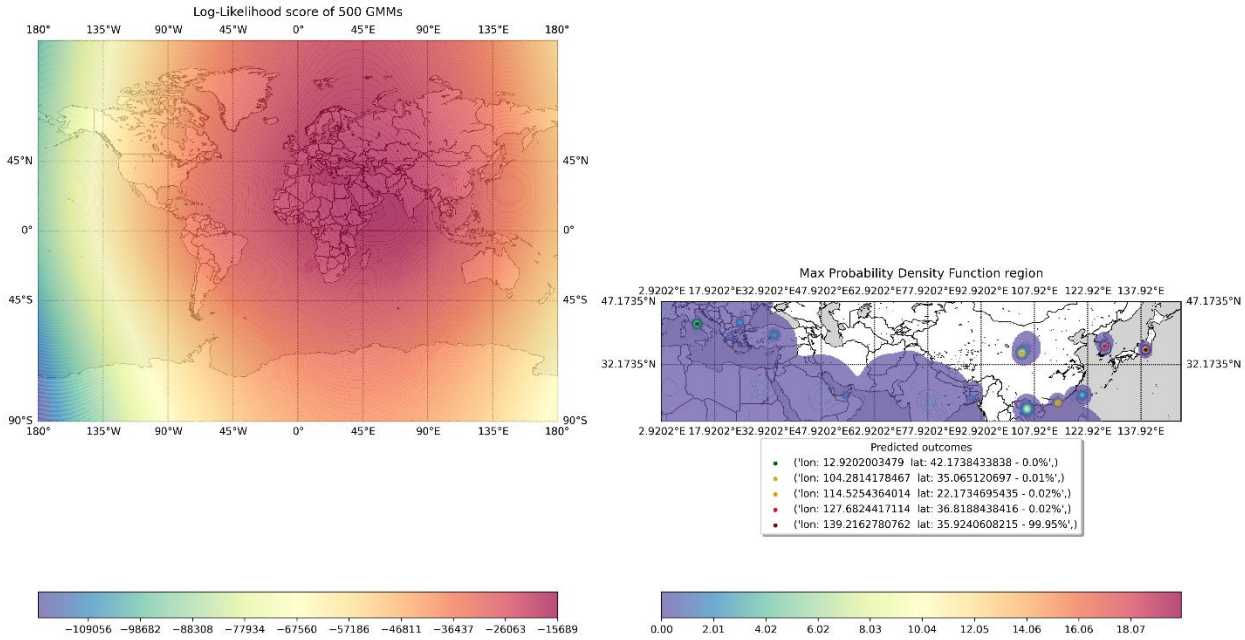
ДОДАТОК В

Візуалізація прогнозу місцезнаходження за текстом



Приклади прогнозування двох моделей для одного і того ж тексту " CIA and FBI can track anyone, and you willingly give the data away"; обидві моделі були навчені на одному і тому ж світовому наборі даних з ключовими NON-GEO і другорядними GEO-ONLY вхідними змінними тексту, кількість результатів (точок) прогнозування – 5;

вище – значущі (за вагою) точки GMOP як діаграми розсіювання; нижче – значущі (за вагою) гаусові піки PMOP як діаграми LLH і PDF



Прогноз для користувача Daily News на основі 500 твітів у вигляді GMM з 5 центрами як локальними максимумами сумарного прогнозу користувача (поверхня середньої PDF для координатної сітки з піків розподілів)

ДОДАТОК Г

Оцінка якості ПЗ за метриками точності прогнозування місцезнаходження

Dataset	Tweets		Users		Scope
	Train	Test Tweet/User	Train	Test Tweet/User	
GeoText	377,616		9,475		US-only
	300K	76K/73K	7,563	1,900/1,800	
Twitter-US	390M		449,649		
Tw-US	16.7M		816,226		
Local	3M	300K/353K	379K	48K/5K	
Twitter-World	12M		1.39M		Worldwide
Tw-World	24M		1,37M		
Local	3M	300K/394K	540K	100K/5K	

Оригінальні та локально відтворені набори даних, розділені на навчальні та тестові підмножини за кількістю твітів та унікальних користувачів; стовпчик "Тест" розділений на характеристики наборів даних для оцінки на один твіт та на одного користувача

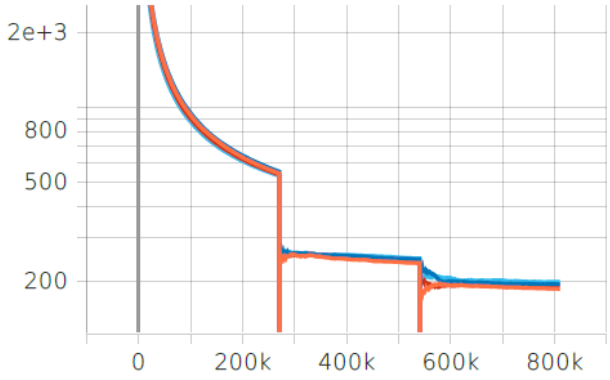
Metric	Priedhorsky [26]	Bakerman [2]	Proposed
Mean CAE	1,445	**557**	600
Med CAE	1,205	117	**79**
Mean $PRA_{0.95}$	3,156,517	**251,120**	15
Med $PRA_{0.95}$	2,846,610	**183,945**	10
$COV_{0.95}$	**99**	94	23

Порівняння результатів імовірнісних метрик у суміжних роботах та запропонованої моделі, навченої та оціненої на наборі даних Twitter-World з використанням гібридного підходу (Контент + Контекст) та виведенням 5 результатів прогнозування для оцінки місцезнаходження твіту та домашнього розташування користувача. Середнє та медіана CAE в км, середнє та медіана $PRA_{0.95}$ в км^2, $COV_{0.95}$ у відсотках

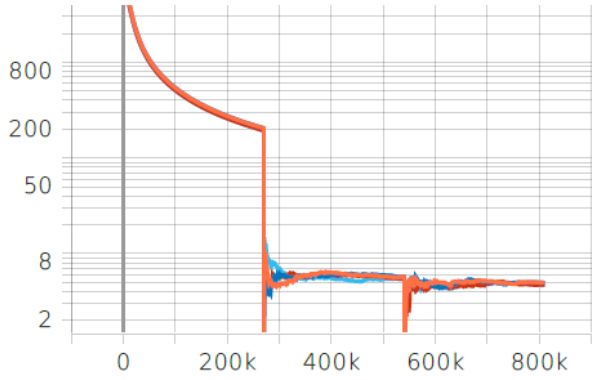
TF	Model	VF	OUT	Spatial			Probabilistic				
				Mean SAE	Med SAE	Acc @161	Mean CAE	Med CAE	Mean $PRA_{0.95}$	Med $PRA_{0.95}$	$COV_{0.95}$
NG+GO	PSOP	TO	1	1881.2	153	50.5	1954	352.8	174	60.8	12.5
		NG		568.3	32.1	78.3	639.2	70.4	60.9	**3.7**	19.6
	PMOP	TO	3	1876.2	134.9	51.5	1911.8	219.9	24.1	16.2	22
		NG		561.7	29.5	78.7	601.1	**63.3**	**13.9**	7.9	**31.2**
		TO	**5**	1845	135.9	51.5	1896.3	214.7	27.9	15.2	15.9
		NG		**551.4**	29.4	**79**	**600.4**	79.1	15.3	9.6	23.4
		TO	5u*	2036.7	234.8	45.1	2080.9	357.5	27.9	28.8	6.2
		NG		626.7	70	70.7	691	158.5	21.3	18.5	9.7
		TO	10	1845	143.2	51.1	1901.4	214.2	32.4	13.9	7.8
		NG		553.2	33.2	78.9	599.8	77.9	15.6	9.4	11
		TO	50	1855.9	136.6	51.4	1905	214.7	27.8	14	1.3
		NG		556.9	29.4	**79**	604.4	77.1	14.7	9.5	2.1
		TO	100	1882.4	149.8	50.6	1939	199.6	193.2	4.5	13.9
		NG		568.9	**28.1**	78.6	618.3	71.1	15.2	9.1	1.2
	GSOP	TO	1	1872.2	140.3	51.3					
		NG		559.6	36.6	78.4					
	GMOP	TO	3	1859.7	142.8	51.1					
		NG		556.3	36.5	78.5			-		
		TO	5	1986.6	151.3	50.6					
		NG		577.8	35.6	78.6					
A	PMOP	TO	5	3203	585.9	41.2	3225.9	623.4	29.6	8.5	7.4
		NG		1266.2	37.7	67	1292.6	62.6	14.7	7.5	11.7
NG		TO		1875.4	172.9	49.3	1927.3	235.2	24.6	13.8	7.5
		NG		581	46.9	76.4	635.2	107	15.9	11.8	12.6
TO		TO		1547.3	176.5	48.7	1708.8	442.1	58.8	38.2	8.8
		NG		782	87.2	64.9	913.1	267.1	36.2	26.4	11.8

Світовий набір даних (300,000 твітів) – результати показників ефективності моделей; TF – змінна навчання; VF – змінна оцінки;

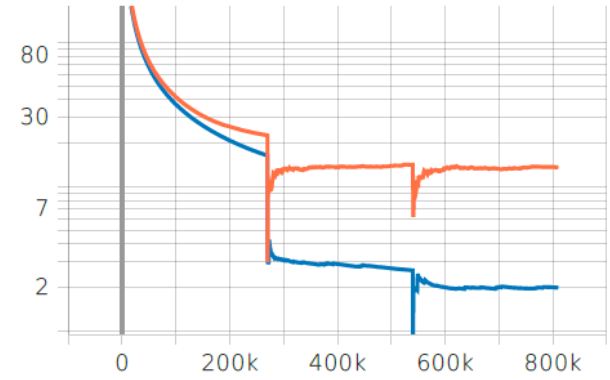
NG+GO : NON-GEO + GEO-ONLY, A : ALL, NG : NON-GEO, TO : TEXT-ONLY; u* – необмежений коваріаційний вихід, $\sigma_{\hat{c}}$ отримано шляхом застосування рівняння (SoftPlus) для виведення параметра коваріації



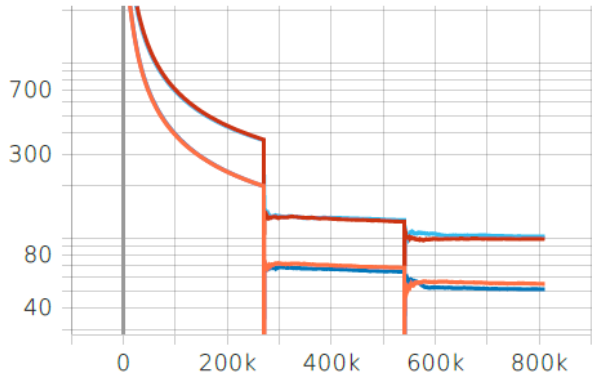
(a) KF Geospatial loss calculated by Eq. (SED)



(b) MF Geospatial loss calculated by Eq. (SED)



(c) Probabilistic loss calculated by Eq. (NLLH)



(d) Total loss calculated as average

Втрати, зареєстровані під час навчання запропонованих моделей з одним та кількома (5) результати та геопросторові та імовірнісні втрати; KF – ключова змінна, MF – другорядна змінна; лінії модельного графіка позначені кольором: помаранчевий – PMOP, синій – PMOP, червоний – GSOP, блакитний – GMOP

ДОДАТОК Д

Формили прогнозування домашнього місцезнаходження користувача

- CAE - *comprehensive accuracy error* metric measures the expected distance between the true origin of the tweet and a random point generated from the GMM predicted by the model. The Haversine formula is used to calculate distances from the original point to each Gaussian sub-population of 100 samples. Bakerman in his work [2] suggests the Monte Carlo method to compute CAE as the mean of the sub-population distance errors:

$$CAE_{SOP} \approx \frac{1}{|z|} \cdot \sum_{\hat{\mathbf{Y}} \in z} Hav(\mathbf{Y}, \hat{\mathbf{Y}})$$

where z is a sample from the Gaussian sub-population. The total CAE of the GMM is calculated as the weighted linear combination of all M Gaussian peaks:

$$CAE_{MOP} \approx \sum_{i=1}^{M} \frac{W_i}{|z_i|} \cdot \sum_{\hat{\mathbf{Y}} \in z_i} Hav(\mathbf{Y}, \hat{\mathbf{Y}})$$

- PRA_α - *prediction region area* metric is the area covered by $\alpha \cdot 100\%$ density of the predicted GMM:

$$PRA_{\alpha,SOP} = \pi \chi_2^2 (1-\alpha) \sigma_{\hat{c}}$$

where σ is a determinant of the spherical covariance matrix $\mathbf{\Sigma}$ from the predicted GMM peak $N(\hat{\mu}, \mathbf{\Sigma})$. In the case of MOP, It's calculated as the weighted linear combination of such area for each of M Gaussian peaks, which depends mainly on their covariance parameters and weights:

$$PRA_{\alpha,MOP} = \pi \chi_2^2 (1-\alpha) \cdot \sum_{i=1}^{M} W_i \sigma_i$$

- COV_α - *coverage* metric measures the proportion of times the prediction region PRA_α covers the true origin of the tweet. Following Bakerman's approach, the original geographic point $\mathbf{Y}$ is within the boundary of an ellipse with center $\hat{\mu}$ and covariance matrix $\mathbf{\Sigma}$ if:

$$\frac{D}{\sigma} \leq \chi_2^2 (1-\alpha); \quad D = Euclidean(\mathbf{Y}, \hat{\mu})$$

Note that in the MOP case, COV_α is calculated as the overall mean of N dataset samples with M outcomes.

Імовірнісні метрики CAE, PRA та COV оцінки моделей з прогнозом місцезнаходження формату GMM

In this work, the focus was primarily on solving the problem of tweet geolocation prediction, thus there is no direct way to predict user's home location using the proposed approaches. However, the PMOP-type models were leveraged to obtain a set of per-tweet predictions for a single user which was then used to estimate the most probable user's home locations in a form of GMOP-type output. Note that in this case the number of selected location points could vary among users, and their weights were set manually based on the summary scores calculated for each outcome.

The summarizing of PMOP-type output of M outcomes was calculated on the grid of $S \cdot M$ GMM peaks gathered from the user's per-tweet predictions put among points of the ground grid G generated on a Mercator projection map with step 10:

$$\mathbf{T} = \{\widehat{\boldsymbol{\mu}}_{i,j} \mid 1 \leq i \leq S, 1 \leq j \leq M\}; \quad \widehat{\boldsymbol{\mu}}_{i,j} = (\widehat{y}_{lon}, \widehat{y}_{lat}); \quad \mathbf{T} \in \mathbb{R}^{(S \cdot M) \times (S \cdot M)}$$

$$\mathbf{C} = \mathbf{G} \cup \mathbf{T}; \quad \mathbf{C}_{i,j} = (y_{lon,i}, y_{lat,j}); \quad \mathbf{G} \in \mathbb{R}^{36 \times 19}$$

The union $\mathbf{C}$ provided a multi-set of all GMM peaks $\mathbf{T}$ merged with a background of grid points $\mathbf{G}$. The average of all S per-tweet scores was calculated for each grid point $\mathbf{C}_{i,j}$ as its likelihood to fit into a predicted GMM of M peaks defined by their weights W, two-dimensional means $\boldsymbol{\mu}$, and covariance matrices $\boldsymbol{\Sigma}$:

$$summary(\mathbf{C}_{i,j}) = \frac{1}{S} \sum_{s=1}^{S} \sum_{m=1}^{M} W_{s,m} \cdot N(\mathbf{C}_{i,j} \mid \widehat{\boldsymbol{\mu}}_{s,m}, \boldsymbol{\Sigma}_{s,m}); \quad \mathbf{C}_{i,j} \in \mathbf{C}$$

Thus forming a two-dimensional matrix $\mathbf{Z}$ containing the average of S probabilities as values of the summarizing function for all grid points $\mathbf{C}_{i,j}$ such that:

$$Z(i,j) = summary(\mathbf{C}_{i,j}); \quad summary : \mathbf{R}^2 \to \mathbf{R}^2$$

Therefore, $Z(i,j)$ was the function value (summary score) of a two-dimensional matrix at point (i,j), and $M_f Z(i,j)$ was the filtered function value of $Z(i,j)$ using a 10 by 10 maxima filter footprint. The formula took the maximum value of $\mathbf{Z}$ within 10×10 window and assigned it to $M_f Z(i,j)$. This operation could be repeated for every point (i,j) in the matrix $\mathbf{Z}$ to obtain the filtered matrix $M_f Z$:

$$M_f Z(i,j) = max_{p=-4}^{5} max_{q=-4}^{5} Z(i+p, j+q)$$

Then the set of local maxima of $\mathbf{Z}$ was defined as:

$$\mathbf{L}^{user} = \{(i,j) \mid (\mathbf{Z}(i,j) = M_f Z(i,j)) \wedge (i,j) \in \mathbf{C}\}$$

where $\mathbf{C}$ was the multi-set of grid points, and $(i,j) \in \mathbf{C}$ means that (i,j) was a point in the grid $\mathbf{C}$.

Note that the bag of non-unique points $\mathbf{C}$ was reduced to the set of unique local maxima points $\mathbf{L}^{user}$ that could consist of a single coordinate pair or multiple unweighted ones. Assuming that $\mathbf{L}^{user}$ contained multiple location points, the top K most probable user's home locations were obtained as the first K elements of the sorted in descending order summary scores for all points in $\mathbf{L}^{user}$ which is referenced as $\mathbf{Z}^{top-K}$:

$$\mathbf{L}^{top-K} = \{(i,j) \mid (\mathbf{Z}(i,j) \in \mathbf{Z}^{top-K} \wedge (i,j) \in \mathbf{L}_{user}\}; \quad \mathbf{L}^{top-K} \in \mathbb{R}^{K \times 2}$$

Then, the coordinate pairs assigned to location indices (i, j) in the set of estimated locations $\mathbf{L}^{top-K}$ were obtained from the initial grid $\mathbf{C}$ as follows:

$$\mathbf{Y}_k^{user} = \mathbf{C}(\mathbf{L}_k^{top-K}) = (y_{lon,i}, y_{lat,j}); \quad \mathbf{Y}^{user} \in \mathbb{R}^{K\times 2}$$

Moreover, weights of the selected location points in $\mathbf{Y}^{user}$ were estimated by the application of Eq. (SoftMax) to their corresponding summary scores in $\mathbf{Z}^{top-K}$ as follows:

$$W_k^{user} = \frac{e^{\mathbf{Z}_k^{top-K}}}{\sum_{j=1}^{K} e^{\mathbf{Z}_j^{top-K}}}; \quad \sum_{k=1}^{K} W_k^{user} = 1; \quad W_k^{user} \in [0, 1]$$

Although the initial model output format had to be of PMOP-type, the estimated points $\mathbf{Y}^{user}$ and their weights W^{user} formed a result similar to the GMOP-type output. Note that the set of multiple prediction outcomes for the task of user's home location prediction could be formed only in the case $\mathbf{L}^{user}$ had more than one unique location. Otherwise, the estimation would result in a GSOP-type output containing a single coordinate pair Y^{user} which excludes the calculation of W^{user}.

Прогнозування домашнього місцезнаходження користувача у вигляді GMOP використовуючи набір PMOP прогнозів для твітів певного користувача

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Алгоритм та програмне забезпечення для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT

Текст програми

КПІ.ІТ-9116.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ігор БАКЛАН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Катерина ЛУЦАЙ

Київ – 2023

GitHub repository: [geo-twitter]

Файл train.sh

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --constraint=GTX1080Ti
#SBATCH --cpus-per-task=12
#SBATCH --export=NONE
#SBATCH --mem 100G # memory pool for all cores # 32GB for folktables ablation
#SBATCH --time 10-00:00:00 # time (D-HH:MM:SS)
#SBATCH --job-name=world-model
#SBATCH -o world-model.%A_%a.%N.out # STDOUT
#SBATCH -e world-model.%A_%a.%N.err # STDERR
#SBATCH --requeue

unset SLURM_EXPORT_ENV

SEED=${SLURM_ARRAY_TASK_ID:-0}
SEEDSTR=$( printf "%01d" $SEED )

hostname
date

module load python/3.8
module load cuda
source ${HOME}/twitter-env/bin/activate

export OMP_NUM_THREADS=12
export CUDA_VISIBLE_DEVICE=0

cd ${HOME}/geo-twitter/

ARG=( "$@" )
srun --cpu_bind=verbose python -u train_bert.py ${ARG[*]}

date
```

Файл data.sh

```
#!/bin/bash

FILES20="/archive3/group/chlgrp/twitter-collection-2020/twitter-2020-*.txt"
FILES21="/archive3/group/chlgrp/twitter-collection-2021/twitter-2021-*.txt"
FILES22="/archive3/group/chlgrp/twitter-collection-2022/twitter-2022-*.txt"

FILES=( $FILES20 $FILES21 $FILES22 )
total=${#FILES[@]}
echo Total number of files is ${total}
c=0


for f in "${FILES[@]}"
do
    if [ -e ${f} ]
    then
        echo ${f} exists
        stat -L -c "%a %G %U" ${f}
        if [ ! -s ${HOME}/geo-twitter/datasets/world/${f:47:18}.txt ]
        then
        echo filtered file is empty or does not exist
        sbatch collector.sh ${f}
        c=$((c+1))
        else
        echo filtered dataset is already collected
        fi
    else
        echo ${f} does not exist
    fi
done
```

```
echo Total number of files is ${total}
echo Total number of scripts launched is ${c}
```

Файл collector.sh

```
#!/bin/bash
#SBATCH -p defaultp # partition (queue)
#SBATCH -N 1 # number of nodes
#SBATCH -c 4 # number of cpus (cores)
#SBATCH --mem 150G # memory pool for all cores # 32GB for folktables ablation
#SBATCH --time 0-1:59:00 # time (D-HH:MM:SS)
#SBATCH --job-name=clt-data
#SBATCH -o world.%A_%a.%N.out # STDOUT
#SBATCH -e world.%A_%a.%N.err # STDERR
#SBATCH --requeue

SEED=${SLURM_ARRAY_TASK_ID:-0}
SEEDSTR=$( printf "%01d" $SEED )

hostname
date

module load python/3.8
source ~/twitter-env/bin/activate

export OMP_NUM_THREADS=4

ARG=( "$@" )
FILE="${ARG[0]}"

if [ -e ${FILE} ]
then
    echo ${FILE} exists
    stat -L -c "%a %G %U" ${FILE}
    cd ${HOME}/geo-twitter/
    srun python -u data-collector.py ${ARG[*]}
else
    echo ${FILE} does not exist
fi

date
```

Файл data-collector.py

```
import json
import sys
import pandas as pd
import os, glob

country_codes = ["CA", "GB", "FR"]

def parse_geo(obj):
    print(obj['coordinates']["coordinates"])
    coord_long = obj['coordinates']["coordinates"][0]
    coord_lat = obj['coordinates']["coordinates"][1]
    return coord_long,coord_lat


def parse_user(obj):
    location = obj['user']['location'] if obj["user"]["location"] else ""
    username = obj['user']['name'] if obj["user"]["name"] else ""
    screen = obj['user']['screen_name'] if obj["user"]["screen_name"] else ""
    description = obj['user']['description'] if obj["user"]["description"] else ""
    user = f"{username} {screen} {description} {location}"
    return user


def parse_place(obj):
    full = obj['place']['full_name'] if obj['place']['full_name'] else ""
    country = obj['place']['country'] if obj['place']['country'] else ""
    code = obj['place']['country_code'] if obj['place']['full_name'] else ""
    name = obj['place']['name'] if obj['place']['country_code'] else ""
```

```
    type = obj['place']['place_type'] if obj['place']['place_type'] else ""
    place = f"{country} {type} {name} {full} {code}"
    return place, code


def parse_tweet(obj):
    text = obj['text']
    time = obj['created_at']
    lang = obj['lang'] if obj['lang'] else ""
    return text, time, lang


def parse_train(fid):
    known_ids = set()

    for line in fid:
        if not line:
          continue
        try:
          obj = json.loads(line)
        except (json.decoder.JSONDecodeError, TypeError):
          print("ERROR: entry wasn't a dictionary. skipping.", file=sys.stderr)
          continue

        try:
          if 'id_str' not in obj:
            print("ERROR: 'id' field not found in tweet", file=sys.stderr)
            continue
          if 'place' not in obj:
            print("ERROR: 'place' field not found in tweet", file=sys.stderr)
            continue
          if 'user' not in obj:
            print("ERROR: 'user' field not found in tweet", file=sys.stderr)
            continue
          if 'coordinates' not in obj:
            print("ERROR: 'coordinates' field not found in tweet", file=sys.stderr)
            continue
          if 'created_at' not in obj:
            print("ERROR: 'created_at' field not found in tweet {}".format(tweet['id']), file =
sys.stderr)
            continue

        except TypeError:
            print("ERROR: not a dict?", line, obj, file=sys.stderr)
            continue

        if not obj["coordinates"]:
            continue
        if not obj["coordinates"]["coordinates"]:
            continue
        if not obj["place"]:
            continue
        # if obj["place"]["country_code"] not in country_codes:
        #     continue
        if not obj["user"]:
            continue

        if obj['id_str'] in known_ids: # duplicate
          continue

        text, time, lang = parse_tweet(obj)
        long, lat = parse_geo(obj)
        place, code = parse_place(obj)
        user = parse_user(obj)
        known_ids.add(id)

        yield (long, lat, text, time, lang, code, place, user)


def read_train(filename):
    print("Reading data from:", filename)
    with open(filename, encoding='utf-8') as fid:
      lines = parse_train(fid)
      longs, lats, texts, times, langs, codes, places, users = zip(*lines)

    data_geo = {
```

```
        'lon':longs,
        'lat':lats,
        'time':times,
        'texts':texts,
        'lang':langs,
        'code':codes,
        'place':places,
        'user':users
    }

    print("Training set of ===", len(longs), "=== samples is collected")

    return pd.DataFrame(data_geo)


def write_records(file, df):
    with open(file, "w") as f:
        df.to_json(f, orient='records', lines=True)
    print("Data written to file:", file)


def combine(file):
    os.chdir(os.path.dirname(__file__) + r"/filtered_json")

    extension = 'txt'
    all_filenames = [i for i in glob.glob('*.{}'.format(extension))]

    combined_txt = pd.concat([pd.read_json(path_or_buf=f, lines=True) for f in all_filenames ])

    #os.chdir(os.path.dirname(__file__))

    with open(file, "w") as f:
        combined_txt.to_json(file, orient='records', lines=True)
    print(f"Data from {len(all_filenames)} files written to common dataset: {file}")


test_input_file = '/run/user/1005618/gvfs/smb-share:server=archive3.ist.local,share=group/chlgrp/twitter-
collection-2022/twitter-2022-01-25.txt'
output_folder = "datasets/"


def main(concat_to_one=False):
    try:  # 1 arg - data input (txt)
        filename = sys.argv[1]
    except IndexError:
        filename = test_input_file
    print(f"Input file: {filename}")

    try:  # 2 arg - filtered data output (jsonl)
        output_filename = sys.argv[2]
    except IndexError:
        head, tail = os.path.split(filename)
        output_filename = output_folder + tail
        open(output_filename, 'w').close()  # if not exists
    print(f"Output file: {output_filename}")

    # manual testing
    # df_geo = read_train(filename)
    # write_records(output_filename, df_geo)

    try:
        df_geo = read_train(filename)
        write_records(output_filename, df_geo)
    except Exception as e:
        print("Couldn't form dataset:", e)

    if concat_to_one:  # if all .txt per-day files are parsed - combine to single json
        combine("ca-twitter-2022.jsonl")


if __name__ == "__main__":
    main()
```

Файл twitter_dataset.py

```
import numpy as np
import pandas as pd

import torch
from torch.utils.data import TensorDataset, DataLoader

import string
import re
import os
import datetime

from sklearn.model_selection import train_test_split

import GPUtil
import psutil

# dataset wrapper


def load_jsonl(filename):
    filename = f"datasets/{filename}"
    print(f"DATASET\tLOAD: {psutil.virtual_memory().percent}%\tLoading dataset from {filename}")
    data = pd.read_json(path_or_buf=filename, lines=True)
    print(f"DATASET\tLOAD: {psutil.virtual_memory().percent}%\tDataset of {len(data.index)} samples and
{len(data.user.unique())} users is loaded")
    return data


def save_df(df, filename, prefix=None):
    if prefix is None:
        prefix = "td-"

    size = len(df.index)
    save_filename = f"datasets/{prefix}{size}-{filename}"

    with open(save_filename, "w") as f:
        df.to_json(f, orient='records', lines=True)
    print(f"DATASET\tSAVE\tTwitter Dataset of {size} samples is written to file: {save_filename}")


def combine_datasets(file_list, filename, prefix=None):
    if prefix is None:
        prefix = "td-"

    df  = load_jsonl(file_list[0])
    for file in file_list[1:]:
        data = load_jsonl(file)
        df = df.append(data, ignore_index=True)

    save_df(df, filename, prefix)


def filter_bots(data, min_total=1, max_day=20):
    print(f"DATASET\tFiltering dataset of {len(data['user'].unique())} users from bots posting more than
{max_day} tweets per day")
    if "time" in data.columns:
        data['date'] = pd.to_datetime(data['time'], utc=False).dt.date
        user_tweets_per_day = data.groupby(['date'])['user'].value_counts()
        user_tweets = user_tweets_per_day[user_tweets_per_day <
max_day].droplevel(0).groupby(["user"]).sum()
    else:
        user_tweets = data['user'].value_counts()

    # data["time"] = data['time'].apply(lambda x: datetime.datetime.combine(x,
datetime.time.min).timestamp())
    data['date'] = data['time'].apply(lambda x: datetime.datetime.strptime(x, '%a %b %d %H:%M:%S %z
%Y').timestamp())
    # data.drop("date", axis=1, inplace=True)
    data["date"] = data["date"].astype(float)

    user_list = user_tweets[user_tweets > min_total].index.tolist()
    data = data[data['user'].isin(user_list)]
```

```
    print(f"DATASET\tSize of the filtered dataset with {len(data['user'].unique())} users:
{len(data.index)} samples")
    return data


# text preprocessing
def nlp_filtering(text):
    def filter_punctuation(text):
        punctuationfree="".join([i for i in text if i not in string.punctuation])
        return punctuationfree

    def filter_websites(text):
        #pattern = r'(http\:\/\/|https\:\/\/)?([a-z0-9][a-z0-9\-]*\.)+[a-z][a-z\-]*'
        pattern = r'http\S+'
        text = re.sub(pattern, '', text)
        return text

    text = filter_websites(text)
    text = filter_punctuation(text)
    return text


def create_dataloader(inputs, masks, labels, batch_size, shuffle=False):
    dataset = TensorDataset(torch.tensor(inputs), torch.tensor(masks), torch.tensor(labels))
    return DataLoader(dataset, batch_size=batch_size, shuffle=shuffle)


# crop dataset to remove already used data before random sampling
def crop_dataset(data, size, seed, by_user=False, ref_file=None, save=False):
    print(f"DATASET\tCropping dataset of {len(data.index)} in {size} samples with seed {seed}{' including
uniqie users' if by_user else ''}")
    if by_user:
        if ref_file:
            df = load_jsonl(ref_file)
            crop_data = df.sample(n=size, random_state=seed)
        else:
            crop_data = data.sample(n=size, random_state=seed)
            data = data.drop(crop_data.index, axis=0)

        crop_users = crop_data["user"].unique()

        print(f"DATASET\tUnique users to crop: {len(crop_users)}")
        data = data[-data["user"].isin(crop_users)]
        print(f"DATASET\tUnique users left: {len(data['user'].unique())}")
    else:
        crop_data = data.sample(n=size, random_state=seed)
        data = data.drop(crop_data.index, axis=0)

    if save and ref_file:
        save_df(crop_data, ref_file, "train-")

    print(f"DATASET\tReduced dataset lenght is {len(data.index)}")
    return data


# sample users (bots filtering) for evaluation
def sample_users(data, users_n, seed=42):
    user_tweets = data['user'].value_counts()
    user_list = user_tweets.sample(n=users_n, random_state=seed).index.tolist()
    return data[data['user'].isin(user_list)]


class TwitterDataloader():
    def __init__(self, filename, features, target, tokenizer, seed=42, scaled=False, val_feature=None,
bot_filter=False):
        self.filename = filename
        self.data = load_jsonl(self.filename)
        if "texts" in self.data.columns:
            self.data.rename(columns={'longitude': 'lon',
                                      'latitude': 'lat',
                                      'created_at': 'time',
                                      'texts': 'text'}, inplace=True)
            self.data.to_json(f"datasets/{self.filename}", orient='records', lines=True)
            print(self.data.info())

        if bot_filter:
```

```
            self.data = filter_bots(self.data)
            save_df(self.data, self.filename, "filtered-")

        self.target = target

        self.tokenizer = tokenizer
        self.seed = seed

        self.scaled = scaled

        self.feature_columns = {
            "GEO": ["text", "place", "user"],
            "NON-GEO": ["text", "user"],
            "NON-USER": ["text", "place"],
            "META-DATA": ["place", "user"],
            "TEXT-ONLY": ["text"],
            "GEO-ONLY": ["place"],
            "USER-ONLY": ["user"],
        }

        self.features = features
        self.n_features = len(features)
        self.key_feature = features[0]
        self.minor_features = features[1:]

        self.val_feature = features[0] if val_feature is None else val_feature

        self.val_dataloader, self.train_dataloader, self.test_dataloader = None, None, None
        self.val_df = None

    # filtering dataset files by column condition
    def filter_dataset(self, column_name, filter_text=None, filter_list=None, save=True):
        if filter_text:
            filter_df = self.data[self.data[column_name] == filter_text]
            prefix = f"f-{column_name}-{filter_text}-td-"
        elif filter_list:
            filter_df = self.data[self.data[column_name].isin(filter_list)]
            prefix = f"f-{column_name}-{'+'.join(filter_list)}-td-"

        if save:
            save_df(filter_df, self.filename, prefix)

    # tokenization - text to IDs and attention masks
    def tokenize(self, column):
        encoded_corpus = self.tokenizer(text=column,
                                        add_special_tokens=True,
                                        padding='max_length',
                                        truncation='longest_first',
                                        max_length=300,
                                        return_attention_mask=True)

        input_ids = encoded_corpus['input_ids']
        attention_masks = encoded_corpus['attention_mask']
        return input_ids, attention_masks

    # forming feature columns, dropping old columns
    def feature_split_filter(self, data):
        text_features = self.features + [self.val_feature] if self.val_feature not in self.features else
self.features
        for f in text_features:
            data[f] = data[self.feature_columns[f]].astype(str).agg(" ".join, axis=1) if
len(self.feature_columns[f]) > 1 else data[self.feature_columns[f]]
            data[f] = data[f].astype(str).apply(nlp_filtering)

        for f in text_features:
            for column in self.feature_columns[f]:
                if column in data.columns:
                    data = data.drop(columns=[column], axis=1)
        return data

    def form_training(self, batch_size, size, test_ratio, skip_size=0, shuffle=True):
        if skip_size > 0:
            self.data = crop_dataset(self.data, skip_size, self.seed)

        print(f"DATASET\tForming training dataset of {size} samples with test size {int(test_ratio*size)}
for features: {', '.join(self.features)}")
        train_df = self.data.sample(n=size, random_state=self.seed).copy()
```

```
        del self.data
        train_df = self.feature_split_filter(train_df)
        self.create_feature_dataloaders(train_df, True, batch_size, shuffle, test_ratio)

    def form_validation(self, batch_size, size, by_user=False, skip_size=0, ref_train_file=None):
        if skip_size > 0:
            self.data = crop_dataset(self.data, skip_size, self.seed, by_user, ref_train_file, True)

#         self.data = filter_bots(self.data)
        save_df(self.data, self.filename, "test-")

        print(f"DATASET\tForming validation dataset of {size} {'users' if by_user else 'samples'} with
batch size {batch_size} for {self.val_feature} text feature")
        if by_user:
            self.val_df = sample_users(self.data, size, seed=self.seed).copy()
            self.features += ["USER-ONLY"]
        else:
            self.val_df = self.data.sample(n=size, random_state=self.seed).copy()
        del self.data

        print(f"DATASET\tSize of the validation dataset with {len(self.val_df['user'].unique())} users:
{len(self.val_df.index)} samples")

        self.val_df = self.feature_split_filter(self.val_df)
        self.create_feature_dataloaders(self.val_df, False, batch_size)

    # training and evaluation dataloaders formation
    def create_feature_dataloaders(self, df, train, batch_size, shuffle=False, test_ratio=None):
        if train:
            train_index, test_index = train_test_split(df.index, test_size=test_ratio,
random_state=self.seed)
            train_index, test_index = list(train_index), list(test_index)
            train_size, test_size = len(train_index), len(test_index)
            train_inputs, train_masks = np.empty((train_size, 0)), np.empty((train_size, 0))

            for feature in self.features:
                input_ids, attention_mask = self.tokenize(df.loc[train_index, feature].tolist())
                train_inputs, train_masks = np.concatenate((train_inputs, input_ids), axis=1),
np.concatenate((train_masks, attention_mask), axis=1)

            test_inputs, test_masks = self.tokenize(df.loc[test_index, self.val_feature].tolist())
            if self.scaled:
                train_labels = np.reshape(np.multiply(df.loc[train_index, self.target].to_numpy(), 0.01),
(train_size, 2))
                test_labels = np.reshape(np.multiply(df.loc[test_index, self.target].to_numpy(), 0.01),
(test_size, 2))
            else:
                train_labels = np.reshape(df.loc[train_index, self.target].to_numpy(), (train_size, 2))
                test_labels = np.reshape(df.loc[test_index, self.target].to_numpy(), (test_size, 2))

            self.train_dataloader = create_dataloader(np.reshape(train_inputs, (train_size,
self.n_features, 300)),
                                                      np.reshape(train_masks, (train_size,
self.n_features, 300)), train_labels, batch_size, shuffle)
            self.test_dataloader = create_dataloader(test_inputs, test_masks, test_labels, batch_size,
shuffle)
            del df
        else:
            labels = df[self.target].to_numpy()
            if self.scaled:
                labels = np.multiply(labels, 0.01)
            val_inputs, val_masks = self.tokenize(df[self.val_feature].tolist())

            self.val_dataloader = create_dataloader(val_inputs, val_masks, labels, batch_size, shuffle)
```

Файл train_bert.py

```
import argparse
from transformers import BertTokenizer
from utils.model_trainer import *

# Entry point for training and evaluation of the models

f = ["GEO", "NON-GEO", "META-DATA", "TEXT-ONLY", "GEO-ONLY", "USER-ONLY"]
```

```
dataset_file = "test-3877395-filtered-16219676-us-twitter-2021.jsonl"  # .jsonl
features = [f[1], f[4]]
val_f = f[1]  # None -> features[0]
target_columns = ["lon", "lat"]

original_worldwide_model = "bert-base-multilingual-cased"
original_usa_model = "bert-base-cased"

# parameters = dict(
#     max_lr = [5e-5, 1e-5],
#     min_lr = [5e-6, 1e-6, 1e-8, 1e-16],
#     scheduler = ["cosine", "plateau"]
# )
# param_values = [v for v in parameters.values()]

covariance_types = [None, "spher"]  # [None, "full", "spher", "diag", "tied"]
scheduler_types = ["cosine", "linear", "plateau"]  # ["cosine", "linear", "cosine-long", "plateau",
"step", "multi step", "one cycle", "cyclic"]

loss_distance = True
loss_mf = "mean"  # mean/sum - mean if features > 1
loss_prob = "pos"  # all/pos - pos if prob
loss_total = "mean"  # sum/mean/type - mean if prob else type (spat)

outcomes = 5
covariance = covariance_types[1]  # None/spher

epochs = 3
log_step = 1000

batch_size = 4

lr_max = 1e-5
lr_min = 1e-6
scheduler = scheduler_types[0]

val_size = 1000  # samples/users if -vu
threshold = 100

train_size = 0
test_ratio = 0.1
seed = 42

ref_file = None #  "us-twitter-2020.jsonl"  # if not None exclude found users from the current data
bot_filter = False


def main():
    parser = argparse.ArgumentParser(description='Finetune multilingual transformer model')
    parser.add_argument('-n', '--nepochs', type=int, default=epochs, help='Number of epochs to train')
    parser.add_argument('-ss', '--skip', type=int, default=0, help='Number of dataset samples to skip')

    parser.add_argument('-sc', '--scale_coord', action="store_true", help="Keep coordinates unscaled
(default: True)")
    parser.add_argument('-o', '--outcomes', type=int, default=outcomes, help="Number of outcomes (lomg,
lat) per tweet")
    parser.add_argument('-c', '--covariance', type=str, default=covariance, help="Covariance matrix type")
    parser.add_argument('-nw', '--weighted', action="store_false", help="Weights of GMM are not equal
(default: True)")

    parser.add_argument('-ld', '--loss_dist', action="store_false", help="Distance loss criterion
(default: True)")
    parser.add_argument('-lmf', '--loss_mf', type=str, default=loss_mf, help="Multi feature loss handle
mean or sum (default: mean)")
    parser.add_argument('-lp', '--loss_prob', type=str, default=loss_prob, help="Probabilistic loss domain
all or pos (default: all)")
    parser.add_argument('-lt', '--loss_total', type=str, default=loss_total, help="Total loss handle by
model type or sum (default: type)")

    parser.add_argument('-m', '--local_model', type=str, default=None, help='Filename prefix of local
model')
    parser.add_argument('--nockp', action="store_false", help='Saving model checkpoints during training
(preset: True)')

    parser.add_argument('-lr', '--learn_rate', type=float, default=lr_max, help='Learning rate (default:
4e-5)')
```

```
    parser.add_argument('-lrm', '--learn_rate_min', type=float, default=lr_min, help='Learning rate
minimum (default: 1e-8)')
    parser.add_argument('-sdl', '--scheduler', type=str, default=scheduler, help="Scheduler type")

    parser.add_argument('-b', '--batch_size', type=int, default=batch_size, help='Per-device batch size
(default: 22)')
    parser.add_argument('-ls', '--log_step', type=int, default=log_step, help='Log step (default: 1000)')

    parser.add_argument('-us', '--usa_model', action="store_true", help="Use USA model instead of
worldwide (default: False)")
    parser.add_argument('-d', '--dataset', type=str, default=dataset_file, help="Input dataset (in jsonl
format)")
    parser.add_argument('-f', '--features', default=features, nargs='+', help="Features names")
    parser.add_argument('-ts', '--train_size', type=int, default=train_size, help='Training dataloader
size')
    parser.add_argument('-tr', '--test_ratio', type=float, default=test_ratio, help='Training dataloader
test ratio (default: 0.1)')
    parser.add_argument('-s', '--seed', type=int, default=seed, help='Random seed (default: 42)')
    parser.add_argument('-v', '--val_size', type=int, default=val_size, help='Validation dataloader size')
    parser.add_argument('-th', '--threshold', type=int, default=threshold, help='Validation threshold in
km (default: 200)')
    parser.add_argument('-vu', '--val_user', action="store_true", help="Form validation dataset by user
(default: False)")

    parser.add_argument('--train', action="store_true", help="Start pretraining")
    parser.add_argument('--eval', action="store_true", help="Start evaluation")
    parser.add_argument('--hptune', action="store_true", help="Start training with hyper parameters
tuning")
    args = parser.parse_args()

    if args.local_model is None:
        prefix = f"{'US-' if args.usa_model else ''}{'U-' if not args.scale_coord else
''}{'+'.join(args.features)}-O{args.outcomes}-{'d' if  args.loss_dist else 'c'}-" \
                 f"total_{args.loss_total if args.covariance is not None else 'type'}-{'mf_' +
args.loss_mf + '-' if len(args.features) > 1 else ''}" \
                 f"{args.loss_prob + '_' if args.covariance is not None else ''}{args.covariance if
args.covariance is not None else 'NP'}-" \
                 f"{'weighted-' if args.weighted and args.outcomes > 1 else
''}N{args.train_size//100000}e5-" \
                 f"B{args.batch_size}-E{args.nepochs}-{args.scheduler}-
LR[{args.learn_rate};{args.learn_rate_min}]"
    else:
        prefix = args.local_model

    print(f"Model prefix:\t{prefix}")
    if torch.cuda.is_available():
        print(f"DEVICE\tAvailable GPU has {torch.cuda.device_count()} devices, using
{torch.cuda.get_device_name(0)}")
        print(f"DEVICE\tCPU has {torch.get_num_threads()} threads")
    else:
        print(f"DEVICE\tNo GPU available, using the CPU with {torch.get_num_threads()} threads instead.")

    original_model = original_usa_model if args.usa_model else original_worldwide_model

#    combine_datasets(["test-10501727-filtered-17174594-worldwide-twitter-2020_0.jsonl", "test-10586286-
filtered-17264575-worldwide-twitter-2020_1.jsonl", "test-3783510-filtered-6464689-worldwide-twitter-
2020_2.jsonl"], "test-filtered-worldwide-twitter-2020.jsonl")

    dataloader = TwitterDataloader(args.dataset,
                                   args.features,
                                   target_columns,
                                   BertTokenizer.from_pretrained(original_model),
                                   args.seed,
                                   args.scale_coord,
                                   val_f,
                                   bot_filter)

#  no settings run to save filtered by condition dataset copy
#    dataloader.filter_dataset("code", "US", None)

    trainer = ModelTrainer(prefix,
                           dataloader,
                           args.nepochs,
                           args.batch_size,
                           args.outcomes,
                           args.covariance,
                           args.weighted,
```

```
                          args.loss_dist,
                          args.loss_mf,
                          args.loss_prob,
                          args.loss_total,
                          args.learn_rate,
                          args.learn_rate_min,
                          original_model)

    # if args.hptune:
    #     trainer.hp_tuning(args.train_size,
    #                       args.test_ratio,
    #                       param_values,
    #                       args.log_step)

    if args.train:
        trainer.finetune(args.train_size,
                         args.test_ratio,
                         f"{prefix}.pth",
                         args.nockp,
                         args.log_step,
                         args.scheduler,
                         args.skip)

    if args.eval:
        trainer.eval(args.val_size,
                     args.threshold,
                     args.val_size,
                     args.val_user,
                     args.train_size,
                     ref_file)


if __name__ == "__main__":
    main()
```

Файл model_trainer.py

```
import time
from datetime import datetime
from pathlib import Path

from itertools import product

import numpy as np
import pandas as pd
import torch
from transformers import BertModel, get_linear_schedule_with_warmup

from torch.optim.lr_scheduler import CosineAnnealingLR, OneCycleLR, ReduceLROnPlateau, StepLR,
MultiStepLR, CyclicLR

import torch.nn as nn
from torch.optim import AdamW
from torch.nn.utils.clip_grad import clip_grad_norm_
from torch.utils.tensorboard import SummaryWriter

from utils.twitter_dataset import *
from utils.result_manager import *
from utils.cosine_scheduler import *
from utils.regressor import *
from utils.benchmarks import *
from utils.result_visuals import *

# model training + evaluation + hp tuning


class ModelTrainer():
    def __init__(self, file_prefix, twitter_dataloader, epochs, batch, outcomes=1, covariance=None,
weighted=False,
                 loss_dist=True, loss_mf="mean", loss_prob="pos", loss_total="sum",
                 learn_rate_max=4e-5, learn_rate_min=1e-8, original_model=None):
        self.data = twitter_dataloader
        self.features = self.data.features
        self.n_features = self.data.n_features
        self.key_feature = self.data.key_feature
```

```
        self.minor_features = self.data.minor_features

        self.scaled = self.data.scaled

        self.val_feature = self.data.val_feature

        self.original_model = original_model
        self.model_file = f"{file_prefix}.pth"

        self.epochs = epochs
        self.batch_size = batch
        self.lr_max = learn_rate_max
        self.lr_min = learn_rate_min

        self.prefix = file_prefix

        if torch.cuda.is_available():
            #print(f"DEVICE\tAvailable GPU has {torch.cuda.device_count()} devices, using
{torch.cuda.get_device_name(0)}")
            self.device = torch.device("cuda")
            self.cluster = True
        else:
            #print(f"DEVICE\tNo GPU available, using the CPU with {torch.get_num_threads()} threads
instead.")
            self.device = torch.device("cpu")
            self.cluster = False

        self.outcomes = outcomes
        self.covariance = covariance
        self.weighted = weighted if self.outcomes != 1 else False
        self.prob = self.covariance is not None
        self.model_folder = "prob" if self.prob else "spat"

        bert_wrapper = BERTregModel(self.outcomes, self.covariance, self.weighted, self.features,
original_model)
        self.model = bert_wrapper.model.to(self.device)
        self.optimizer = AdamW(self.model.parameters(), lr=self.lr_max, eps=1e-8)

        self.key_outputs = bert_wrapper.key_output
        self.minor_outputs = bert_wrapper.minor_output

        loss_total = "type" if not self.prob else loss_total
        self.benchmark = ModelBenchmark(bert_wrapper, loss_dist, loss_prob, loss_mf, loss_total)

        self.key_loss = self.benchmark.key_feature_loss
        self.r2_score = self.benchmark.r2
        self.result_metrics = self.benchmark.result_metrics
        self.minor_loss = None if self.n_features == 1 else self.benchmark.minor_feature_loss
        if self.prob:
            self.prob_models = self.benchmark.prob_models

    def load_local_model(self, local_model=None):
        if local_model is None:
            local_model = self.model_file
        local_model = f"models/full/{self.model_folder}/{local_model}"
        print(f"LOAD\tLoading model from {local_model}")

        if not Path(local_model).is_file():
            print(f"LOAD [ERROR] Unable to load local model: file {local_model} does not exist")
            return

        if self.cluster:
            state = torch.load(local_model)
        else:
            state = torch.load(local_model, map_location='cpu')

        self.model.load_state_dict(state['model_state_dict'])
        self.optimizer.load_state_dict(state['optimizer_state_dict'])

    def save_local_model(self, local_model=None):
        if local_model is None:
            local_model = self.model_file
        local_model = f"models/full/{self.model_folder}/{local_model}"
        print(f"SAVE\tSaving model to file {local_model}")

        state = {'model_state_dict': self.model.state_dict(),
                 'optimizer_state_dict': self.optimizer.state_dict()}
```

```
        torch.save(state, local_model)

    def load_checkpoint(self, ckp_file=None):
        if ckp_file is None:
            ckp_file = self.ckp_file
        ckp_file = f"models/ckp/full/{self.model_folder}/{ckp_file}"
        print(f"LOAD\tLoading model checkpoint from {ckp_file}")

        if not Path(ckp_file).is_file():
            print(f"LOAD [ERROR] Unable to load local model checkpoint: file {ckp_file} does not exist")
            return 0, None

        if self.cluster:
            checkpoint = torch.load(ckp_file)
        else:
            checkpoint = torch.load(ckp_file, map_location='cpu')

        self.model.load_state_dict(checkpoint['model_state_dict'])
        self.optimizer.load_state_dict(checkpoint['optimizer_state_dict'])

        if "train_loss_mean" not in checkpoint:
            train_loss_mean = None
        else:
            train_loss_mean = checkpoint['train_loss_mean']

        if "start_epoch" not in checkpoint:
            starting_epoch = 0
        else:
            starting_epoch = checkpoint['start_epoch']

        return starting_epoch, train_loss_mean

    def save_checkpoint(self, current_epoch, current_loss, ckp_file=None):
        if ckp_file is None:
            ckp_file = self.ckp_file
        ckp_file = f"models/ckp/full/{self.model_folder}/{ckp_file}"
        print(f"SAVE\tSaving model checkpoint to file {ckp_file}")

        checkpoint = {'start_epoch': current_epoch + 1,
                      'train_loss_mean': current_loss,
                      'model_state_dict': self.model.state_dict(),
                      'optimizer_state_dict': self.optimizer.state_dict()}

        torch.save(checkpoint, ckp_file)

    # raw prediction results for spat models
    def predict(self, dataloader):
        self.model.eval()
        output = np.empty((0, self.key_outputs), float)
        for batch in dataloader:
            batch_inputs, batch_masks, _ = tuple(b.to(self.device) for b in batch)
            with torch.no_grad():
                if self.cluster:
                    output = np.append(output, self.model(batch_inputs, batch_masks,
self.key_feature).cpu().numpy(), axis=0)
                else:
                    output = np.append(output, self.model(batch_inputs, batch_masks,
self.key_feature).numpy(), axis=0)
        return output

    #  training per-epoch metrics
    def test_metrics(self, dataloader, model=None):
        model = self.model if model is None else model
        val_metric = np.zeros([len(dataloader), 2, 2], dtype=float)
        model.eval()
        for step, batch in enumerate(dataloader):
            batch_metric = np.zeros((2, 2), dtype=float)
            batch_inputs, batch_masks, batch_labels = tuple(b.to(self.device) for b in batch)

            batch_inputs = batch_inputs.to(torch.int64)
            batch_masks = batch_masks.to(torch.int64)

            if self.cluster:
                torch.cuda.empty_cache()
                batch_inputs, batch_masks, batch_labels = batch_inputs.cuda(), batch_masks.cuda(),
batch_labels.cuda()
```

```
            with torch.no_grad():
                outputs = model(batch_inputs, batch_masks, self.key_feature)

            spat_loss, prob_loss = self.key_loss(outputs, batch_labels)

            batch_metric[0, 0] = spat_loss.item()
            batch_metric[0, 1] = self.r2_score(outputs, batch_labels).item()
            if self.prob:
                batch_metric[1, 0] = prob_loss.item()
                batch_metric[1, 1] = torch.exp(-prob_loss).item()

            val_metric[step, :] = batch_metric.reshape(1, 2, 2)

        return val_metric

    # result metric
    def results(self, dataloader, val_size, model=None):
        model = self.model if model is None else model
        model.eval()

        val_metric = np.zeros([val_size, 2, 2], dtype=float)
        pm = [] if self.prob else None

        for step, batch in enumerate(dataloader):
            current_batch_size = len(batch[0])
            batch_metric = np.zeros((current_batch_size, 2, 2), dtype=float)
            batch_inputs, batch_masks, batch_labels = tuple(b.to(self.device) for b in batch)
            if self.cluster:
                torch.cuda.empty_cache()
                batch_inputs, batch_masks, batch_labels = batch_inputs.cuda(), batch_masks.cuda(),
batch_labels.cuda()
            with torch.no_grad():
                outputs = model(batch_inputs, batch_masks, self.key_feature)

            spat_loss, prob_loss = self.result_metrics(outputs, batch_labels)

            if self.cluster:
                batch_metric[:, 0, 0] = spat_loss.cpu().numpy().flatten()
                batch_metric[:, 0, 1] = self.r2_score(outputs, batch_labels).cpu().numpy().flatten()
                if self.prob:
                    batch_metric[:, 1, 0] = prob_loss.cpu().numpy().flatten()
                    batch_metric[:, 1, 1] = torch.exp(-prob_loss).cpu().numpy().flatten()
            else:
                batch_metric[:, 0, 0] = spat_loss.numpy().flatten()
                batch_metric[:, 0, 1] = self.r2_score(outputs, batch_labels).numpy().flatten()
                if self.prob:
                    batch_metric[:, 1, 0] = prob_loss.numpy().flatten()
                    batch_metric[:, 1, 1] = torch.exp(-prob_loss).numpy().flatten()

            current_slice = step*self.batch_size
            val_metric[current_slice:current_slice+current_batch_size, :] =
batch_metric.reshape(current_batch_size, 2, 2)

            if self.prob:
                pm.append(self.prob_models(outputs))

        return val_metric, pm

    # evaluation entry point
    def eval(self, val_size, threshold=200, map_size=1000, by_user=False, skip_size=0, ref_file=None):
        if map_size > val_size:
            map_size = val_size

        print(f"\nStarting evaluation for model {self.model_file}")
        self.load_local_model()

        self.data.form_validation(self.batch_size, val_size, by_user, skip_size, ref_file)
        val_size = len(self.data.val_dataloader.dataset)

        print(f"\nCalculating result metrics for {val_size} samples")
        val_metric, prob_models = self.results(self.data.val_dataloader, val_size)
        print(f"LOG\tVal mean:\n\tGeospatial:\t{'D^2' if self.benchmark.dist else 'Coord'}
Loss:\t{np.mean(val_metric[:, 0, 0], axis=0)}\tCoord R2:\t{np.mean(val_metric[:, 0, 1], axis=0)}")
        if self.prob:
            print(f"\tProbabilistic:\t-LLH Loss:\t{np.mean(val_metric[:, 1, 0],
axis=0)}\tProbability:\t{np.mean(val_metric[:, 0, 1], axis=0)}")
```

```
        result = ResultManager(self.data.val_df,
                               None,
                               self.val_feature,
                               self.device,
                               self.benchmark,
                               self.scaled,
                               by_user,
                               self.prefix)
        if self.prob:
            result.soft_outputs(prob_models)
        else:
            result.coord_outputs(self.predict(self.data.val_dataloader))

        result.metrics(val_metric)
        result.save_df()

        result.performance()

        visual = ResultVisuals(result)
        visual.density()
        visual.cum_dist(False, threshold)

    # training flow
    def train(self, start_epoch, train_loss_saved, scheduler, writer, log_step, ckp=True, clip_value=2):
        current_tlm = train_loss_saved
        if current_tlm is not None:
            print(f"TRAIN\tMean loss of saved model:\t{current_tlm}")

        start_epoch = int(start_epoch)
        epoch_steps = len(self.data.train_dataloader)
        print(f"TRAIN\tProgress of each epoch with {epoch_steps} steps will be logged every {log_step}th
batch iteration")

        for epoch in range(start_epoch, self.epochs+start_epoch):
            print(f"\nEPOCH\t{epoch+1}\t{datetime.now().strftime('%d/%m/%Y %H:%M:%S')}\tStarting training
...")

            #train_metric = torch.empty((0, self.n_features, 2, 2), device=batch_inputs.device)
            train_metric = np.zeros([epoch_steps, self.n_features, 2, 2], dtype=float)

            self.model.train()
            for step, batch in enumerate(self.data.train_dataloader):
                ts = time.time()
                batch_inputs, batch_masks, batch_labels = tuple(b.to(self.device) for b in batch)

                self.optimizer.zero_grad()
                if self.cluster:
                    torch.cuda.empty_cache()
                batch_loss = torch.zeros((self.n_features, 2), device=batch_inputs.device)
                batch_metric = np.zeros((self.n_features, 2, 2), dtype=float)
                #minor_loss = torch.zeros(len(self.minor_features), device=batch_inputs.device)
                for f in range(len(self.features)):
                    feature = self.features[f]
                    feature_inputs = batch_inputs[:, f, :].to(torch.int64)
                    feature_masks = batch_masks[:, f, :].to(torch.int64)
                    if self.cluster:
                        feature_inputs, feature_masks, batch_labels = feature_inputs.cuda(),
feature_masks.cuda(), batch_labels.cuda()

                    outputs = self.model(feature_inputs, feature_masks, feature)
                    spat_loss, prob_loss = self.key_loss(outputs, batch_labels) if feature ==
self.key_feature else self.minor_loss(outputs, batch_labels)

                    batch_metric[f, 0, 0] = spat_loss.item()
                    if feature == self.key_feature:
                        batch_metric[f, 0, 1] = self.r2_score(outputs, batch_labels).item()
                    if self.prob:
                        batch_metric[f, 1, 0] = prob_loss.item()
                        if feature == self.key_feature:
                            batch_metric[f, 1, 1] = torch.exp(-prob_loss).item()

                    batch_loss[f] = torch.stack((spat_loss, prob_loss))

                train_metric[step, :] = batch_metric.reshape(self.n_features, 2, 2)
                # print(batch_loss)
                # print(torch.mean(batch_loss, dim=0))
```

```
                # print(torch.sum(torch.mean(batch_loss, dim=0), dim=0))
                train_loss = self.benchmark.total_batch_loss(batch_loss)
                # print(train_loss)
                train_loss.backward()

                if (step % log_step) == 0:
                    print(f"EPOCH\t{epoch+1}\t{datetime.now().strftime('%d/%m/%Y %H:%M:%S')}\tStep:
{step+1}/{epoch_steps} took {round(time.time()-ts,2)}s")
                    self.benchmark.log(writer, step + epoch*epoch_steps + 1,
scheduler.optimizer.param_groups[0]["lr"], train_metric, step+1)
                    if isinstance(scheduler, ReduceLROnPlateau):
                        scheduler.step(np.mean(train_loss))

                clip_grad_norm_(self.model.parameters(), clip_value)
                self.optimizer.step()
                if not isinstance(scheduler, ReduceLROnPlateau):
                    scheduler.step()

            print(f"EPOCH\t{epoch+1}\tCalculating evaluation metrics")
            val_metric = self.test_metrics(self.data.test_dataloader)
            self.benchmark.log(writer, (epoch+1)*epoch_steps, scheduler.optimizer.param_groups[0]["lr"],
train_metric, step+1, val_metric)

            current_tlm = self.benchmark.mean_epoch_train_loss

            if ckp:
                self.save_checkpoint(epoch, current_tlm)
                if train_loss_saved is None:
                    self.save_local_model()
                    train_loss_saved = current_tlm
                else:
                    if current_tlm <= train_loss_saved:
                        print(f"EPOCH\t{epoch+1}\tTraining loss decreased: {train_loss_saved} -->
{current_tlm}.  Saving model ...")
                        self.save_local_model()
                        train_loss_saved = current_tlm
                    else:
                        print(f"EPOCH\t{epoch+1}\tTraining loss increased: {train_loss_saved} -->
{current_tlm}")

            else:
                self.save_local_model()

        return current_tlm

    # training entry point
    def finetune(self, train_size, test_ratio, local_model=None, ckp=True, log_step=1000,
scheduler_type="cosine", skip_size=0):
        self.data.form_training(self.batch_size, train_size, test_ratio, skip_size=skip_size)

        if local_model is not None:
            self.model_file = local_model

        self.load_local_model()
        start_epoch = 0
        train_loss_saved = None

        if ckp:
            self.ckp_file = f"ckp-{self.prefix}.pth"
            start_epoch, train_loss_saved = self.load_checkpoint()

        log_folder = f"./runs/full/{self.model_folder}/{self.prefix}-{datetime.today().strftime('%Y-%m-
%d')}_logs"
        Path(log_folder).mkdir(parents=True, exist_ok=True)
        writer = SummaryWriter(log_folder)

        print(f"\nStarting training for {self.epochs} epochs with {len(self.data.train_dataloader) *
self.epochs} steps in total")

        self.train(start_epoch,
                   train_loss_saved,
                   self.get_scheduler(scheduler_type),
                   writer,
                   log_step,
                   ckp=ckp,
                   clip_value=2)
```

```
        print(f"\nTraining is finished:\nBest model is saved to {self.model_file}")
        if ckp is not None:
            print(f"Last model checkpoint is saved to {self.ckp_file}")

        writer.close()

    # hp tuning entry point
    def hp_tuning(self, train_size, test_ratio, param_values, log_step=1000):
        self.data.form_training(self.batch_size, train_size, test_ratio, True)

        epoch_steps = len(self.data.train_dataloader)
        total_steps = epoch_steps * self.epochs

        logs_folder = f"./runs/hptune_logs/full/{self.model_folder}/{datetime.today().strftime('%Y-%m-
%d')}_schedulers"
        Path(logs_folder).mkdir(parents=True, exist_ok=True)

        print(f"\nHyper parameter tuning for {len(list(product(*param_values)))} runs without model saving
and evaluation")

        for run_id, (max_lr, min_lr, scheduler) in enumerate(product(*param_values)):
            bert_wrapper = BERTregModel(self.outcomes, self.covariance, self.weighted)
            model = bert_wrapper.model.to(self.device)
            optimizer = AdamW(model.parameters(), lr=self.lr_max, eps=1e-8)

            run_scheduler = self.get_scheduler(scheduler, max_lr, min_lr, optimizer)

            print(f"\nStarting training for {self.epochs} epochs with {total_steps} steps in total")
            print(f"\tRun ID\t{run_id + 1}\n\tMAX lr\t{max_lr}\n\tMIN
lr\t{min_lr}\n\tscheduler\t{scheduler}")

            run_folder = f"{logs_folder}/SCHEDULER:{scheduler};MAX_LR:{max_lr};MIN_LR:{min_lr}"
            Path(run_folder).mkdir(parents=True, exist_ok=True)
            writer = SummaryWriter(run_folder)

            for epoch in range(self.epochs):
                print(f"\nEPOCH\t{epoch+1}\t{datetime.now().strftime('%d/%m/%Y %H:%M:%S')}\tStarting
training ...")

                spatial_loss, spatial_r2 = [], []
                lh_loss, prob = None, None
                if self.prob:
                    lh_loss, prob = [], []

                model.train()
                for step, batch in enumerate(self.data.train_dataloader):
                    ts = time.time()
                    batch_inputs, batch_masks, batch_labels = tuple(b.to(self.device) for b in batch)
                    if self.cluster:
                        torch.cuda.empty_cache()
                        batch_inputs, batch_masks, batch_labels = batch_inputs.cuda(), batch_masks.cuda(),
batch_labels.cuda()

                    optimizer.zero_grad()
                    minor_loss = torch.zeros(len(self.minor_features), device=batch_inputs.device)
                    for f in range(len(self.features)):
                        feature = self.features[f]
                        feature_inputs = batch_inputs[:, f, :]
                        feature_masks = batch_masks[:, f, :]
                        outputs = model(feature_inputs.to(torch.int64), feature_masks.to(torch.int64),
feature)

                        if feature == self.key_feature:
                            key_loss = self.loss_function(outputs, batch_labels)
                            spatial_loss.append(self.benchmark.spatial_loss(outputs, batch_labels).item())
                            spatial_r2.append(self.benchmark.r2(outputs, batch_labels).item())

                            if self.prob:
                                lh = self.benchmark.log_likelihood_loss(outputs, batch_labels)
                                lh_loss.append(lh.item())
                                prob.append(torch.exp(-lh).item())
                        else:
                            minor_loss[f-1] = self.benchmark.minor_spatial_loss(outputs, batch_labels)

                    total_batch_loss = key_loss + torch.sum(minor_loss)
                    total_batch_loss.backward()
```

```
                    optimizer.step()

                    if (step % log_step) == 0:
                        print(f"EPOCH\t{epoch+1}\t{datetime.now().strftime('%d/%m/%Y %H:%M:%S')}\tStep:
{step+1}/{epoch_steps} took {round(time.time()-ts,2)}s")
                        self.benchmark.log(writer, step + epoch*epoch_steps + 1,
run_scheduler.optimizer.param_groups[0]["lr"],
                                           spatial_loss, spatial_r2, total_batch_loss, lh_loss, prob)
                        if isinstance(run_scheduler, ReduceLROnPlateau):
                            run_scheduler.step(np.mean(lh_loss if self.prob else spatial_loss))

                    clip_grad_norm_(model.parameters(), 2)
                    if not isinstance(run_scheduler, ReduceLROnPlateau):
                        run_scheduler.step()

                print(f"EPOCH\t{epoch+1}\tCalculating evaluation metrics")
                val_spatial_loss, val_spatial_r2, val_lh_loss, val_prob, =
self.test_metrics(self.data.test_dataloader)
                self.benchmark.log(writer, (epoch+1)*epoch_steps,
run_scheduler.optimizer.param_groups[0]["lr"],
                                   spatial_loss, spatial_r2, total_batch_loss, lh_loss, prob,
                                   val_spatial_loss, val_spatial_r2, val_lh_loss, val_prob)

            if self.prob:
                writer.add_hparams(
                    {"max_lr": max_lr, "min_lr": min_lr, "scheduler": scheduler},
                    {"train_spatial_loss": np.mean(spatial_loss), "val_spatial_loss":
np.mean(val_spatial_loss),
                    "train_lh_loss": np.mean(lh_loss), "val_lh_loss": np.mean(val_lh_loss)}
                )
            else:
                writer.add_hparams(
                    {"max_lr": max_lr, "min_lr": min_lr, "scheduler": scheduler},
                    {"train_spatial_loss": np.mean(spatial_loss), "val_loss": np.mean(val_spatial_loss)}
                )
            writer.flush()

        writer.close()

    # scheduler hp tuning choice
    def get_scheduler(self, type, max_lr=None, min_lr=None, optimizer=None):
        if min_lr is None:
            min_lr = self.lr_min

        if max_lr is None:
            max_lr = self.lr_max

        if optimizer is None:
            optimizer = self.optimizer

        koef = 0.5

        epoch_steps = len(self.data.train_dataloader)
        total_steps = epoch_steps * self.epochs
        long_launch = 5

        print(f"TRAIN\tScheduler type:\t{type}\n\t\tMax LR:\t{max_lr}\n\t\tMin LR\t{min_lr}")

        if type == "cosine":
            scheduler = CyclicCosineDecayLR(optimizer,
                                            init_decay_epochs=total_steps,
                                            min_decay_lr=min_lr)
        if type == "cosine-long":
            scheduler = CyclicCosineDecayLR(optimizer,
                                            init_decay_epochs=total_steps*long_launch,
                                            min_decay_lr=min_lr)
        elif type == "cyclic":
            scheduler = CyclicLR(optimizer,
                                 cycle_momentum=False,
                                 base_lr=min_lr,
                                 max_lr=max_lr)
        elif type == "plateau":
            scheduler = ReduceLROnPlateau(optimizer,
                                          patience=2,
                                          mode="min",
                                          min_lr=min_lr)
        elif type == "step":
```

```
            scheduler = StepLR(optimizer,
                               step_size=epoch_steps//5,
                               gamma=koef)
        elif type == "multi-step":
            scheduler = MultiStepLR(optimizer,
                                    milestones=[epoch_steps//5, total_steps-epoch_steps//5],
                                    gamma=koef)
        elif type == "one-cycle":
            scheduler = OneCycleLR(optimizer,
                                   total_steps=total_steps,
                                   max_lr=max_lr)
        elif type == "linear":
            scheduler = get_linear_schedule_with_warmup(optimizer,
                                                        num_warmup_steps=0,
                                                        num_training_steps=total_steps)

        return scheduler
```

Файл benchmarks.py

```
import math
import torch
import torch.distributions as dist
import numpy as np
from utils.regressor import *

# model benchmarks for loss and metrics


# R2 score
def r2_score(X, Y):
    labels_mean = torch.mean(Y)
    ss_tot = torch.sum((Y - labels_mean) ** 2)
    ss_res = torch.sum((Y - X) ** 2)
    r2 = 1 - ss_res / ss_tot
    return r2


# spher cov from raw outputs
def spher_sigma(X, outcomes, outputs_map, lower_limit=0):
    softplus = nn.Softplus()
    S = softplus(X[:, outputs_map["sigma"][0]:outputs_map["sigma"][1]]) + lower_limit
    return S.reshape([X.size(dim=0), outcomes])


# GM/GMM from raw outputs
def GaussianModel(X, outcomes, outputs_map, prob_domain, cov):
    softplus = nn.Softplus()
    batch = X.size(dim=0)

    means = X[:, outputs_map["coord"][0]:outputs_map["coord"][1]]
    if outcomes > 1:
        means = means.reshape([batch, outcomes, 2])

    sigma_lower_limit = 1 / (2 * math.pi) if prob_domain == "pos" else 0
    positive_sigma = spher_sigma(X, outcomes, outputs_map, sigma_lower_limit)

    sigma = None
    tril = None

    if cov == "spher":
        sigma = torch.eye(2, device=X.device) * positive_sigma.reshape(-1, 1)[:, None]
    elif cov == "diag":
        sigma = torch.eye(2, device=X.device) * positive_sigma.reshape(-1, 2)[:, None]
    else:
        tril_indices = torch.tril_indices(row=2, col=2, offset=0, device=X.device)
        if cov == "tied" or outcomes == 1:
            tril = torch.zeros((2, 2), device=X.device).repeat(batch, 1).reshape([batch, 2, 2])
            tril[:, tril_indices[0], tril_indices[1]] = positive_sigma.reshape([batch, 3])
        else:
            tril = torch.zeros((2, 2), device=X.device).repeat(batch, outcomes).reshape([batch, outcomes,
2, 2])
            tril[:, :, tril_indices[0], tril_indices[1]] = positive_sigma.reshape([batch, outcomes, 3])

    if sigma is not None:
        if outcomes > 1:
```

```
            sigma = sigma.reshape([batch, outcomes, 2, 2])
        gaussian = dist.MultivariateNormal(means, sigma)
    else:
        if outcomes > 1 and cov == "tied":
            tril = tril.reshape(batch, -1).repeat(1, outcomes).reshape([batch, outcomes, 2, 2])
        gaussian = dist.MultivariateNormal(means, scale_tril=tril)

    return gaussian


# GMM weights from raw outputs
def GaussianWeights(X, outcomes, outputs_map):
    softmax = nn.Softmax(dim=1)
    weights = X[:, outputs_map["weight"][0]:outputs_map["weight"][1]].reshape([X.size(dim=0), outcomes])
if outputs_map["weight"] else torch.ones((X.size(dim=0), outcomes), device=X.device)
    gmm_weights = dist.Categorical(softmax(weights))
    return gmm_weights


# spatial weights from raw outputs
def weights(X, outcomes, outputs_map):
    softmax = nn.Softmax(dim=1)
    W = X[:, outputs_map["weight"][0]:outputs_map["weight"][1]].reshape([X.size(dim=0), outcomes]) if
outputs_map["weight"] else torch.ones((X.size(dim=0), outcomes), device=X.device)
    return softmax(W)


# Distance^2 to genuine truth from raw outputs
def dist2(X, y, outcomes, outputs_map):
    def coord_se(X, y, outcomes):
        error_by_coord = nn.MSELoss(reduction='none')
        Y = y.repeat(1, outcomes)
        E = error_by_coord(X, Y)
        return E

    E = coord_se(X[:, outputs_map["coord"][0]:outputs_map["coord"][1]], y, outcomes)
    D = torch.zeros((X.size(dim=0), 0), device=X.device)
    for i in range(outputs_map["coord"][0], outputs_map["coord"][1], 2):
        D = torch.cat((D, torch.sum(E[:, i:i+2], dim=1, keepdim=True)), 1)
    return D


# Negative Log Likelihood fit for genuine truth from raw outputs
def lh_loss(X, y, outcomes, outputs_map, prob_domain):
    gaussian = GaussianModel(X, outcomes, outputs_map, prob_domain, "spher")
    if outcomes > 1:
        gmm_weights = GaussianWeights(X, outcomes, outputs_map)
        gmm = dist.MixtureSameFamily(gmm_weights, gaussian)
        L = -gmm.log_prob(y)
    else:
        L = -gaussian.log_prob(y)
    return L


# weighted D2 loss from raw outputs
def d_loss(X, y, outcomes, outputs_map):
    D = dist2(X, y, outcomes, outputs_map)
    if outcomes > 1:
        W = weights(X, outcomes, outputs_map)
        L = torch.sum(D * W, dim=1)
    else:
        L = D
    return L


class ModelBenchmark():
    def __init__(self, model, distance=True, loss_prob="pos", mf_loss="mean", total_loss="type"):
        self.model = model
        self.dist = distance
        self.prob_domain = loss_prob
        self.mf_handle = mf_loss
        self.total_loss_crit = total_loss

        self.outcomes = self.model.n_outcomes
        self.cov = self.model.cov
        self.weighted = self.model.weighted
```

```
        self.features = self.model.features

        self.outputs_map = {
            "coord": [0, self.model.coord_output],
            "weight": [self.model.coord_output, self.model.coord_output + self.model.weights_output] if
self.weighted else None,
            "sigma": [self.model.coord_output + self.model.weights_output, self.model.coord_output +
self.model.weights_output + self.model.cov_output] if self.cov else None
        }

        self.single_outputs_map = {
            "coord": [0, 2],
            "weight": None,
            "sigma": [2, 3] if self.cov else None
        }

        self.loss_type = 1 if self.outputs_map["sigma"] else 0

        print(f"TRAIN\tLOSS\tKey Feature - {'sum of spat and prob' if self.total_loss_crit == 'sum' else
'by model type'}:\n"
              f"\tGeospatial accuracy:\t{'weighted ' if self.weighted else ''}{'distance' if self.dist
else 'coord'} error^2 for {self.outcomes} outcome(s)")

        if self.outputs_map["sigma"] is not None:
            print(f"\tProbability accuracy:\t {'limited' if self.prob_domain == 'pos' else 'unlimited'} -
LLH for PDF of {'weighted ' if self.weighted else ''}"
                  f"{'GM' if self.outcomes == 1 else 'GMM'} with {self.cov} covariance matrix ")

        if len(self.features) > 1:
            print(f"TRAIN\tLOSS\tMinor Features - {self.mf_handle} of:\n\tGeospatial accuracy:\tsingle
{'distance' if self.dist else 'coord'} error^2")
            if self.outputs_map["sigma"]:
                print(f"\tProbability accuracy:\t {'limited' if self.prob_domain == 'pos' else
'unlimited'} -LLH for PDF of single GM with spher covariance matrix ")

    def minor_feature_loss(self, outputs, labels):
        X = outputs.squeeze().float()
        y = labels.squeeze().float()

        if X.dim() == 1:
            X = X.reshape(1, -1)

        spat_loss = d_loss(X, y, 1, self.single_outputs_map).mean()
        prob_loss = torch.zeros_like(spat_loss, device=X.device)
        if self.outputs_map["sigma"]:
            prob_loss = lh_loss(X, y, 1, self.single_outputs_map, self.prob_domain).mean()
        return spat_loss, prob_loss

    def key_feature_loss(self, outputs, labels):
        X = outputs.squeeze().float()
        y = labels.squeeze().float()

        if X.dim() == 1:
            X = X.reshape(1, -1)

        spat_loss = d_loss(X, y, self.outcomes, self.outputs_map).mean()
        prob_loss = torch.zeros_like(spat_loss, device=X.device)
        if self.outputs_map["sigma"]:
            prob_loss = lh_loss(X, y, self.outcomes, self.outputs_map, self.prob_domain).mean()
        return spat_loss, prob_loss

    def total_batch_loss(self, batch_loss):
        all_features_loss = torch.mean(batch_loss, dim=0) if self.mf_handle == "mean" else
torch.sum(batch_loss, dim=0)
        if self.total_loss_crit == "sum":
            total_loss = torch.sum(all_features_loss, dim=0)
        elif self.total_loss_crit == "mean":
            total_loss = torch.mean(all_features_loss, dim=0)
        elif self.total_loss_crit == "type":
            total_loss = all_features_loss[1 if self.outputs_map["sigma"] else 0]
        return total_loss

    # pytorch GM/GMM from raw outputs
    def prob_models(self, outputs):
        X = outputs.squeeze().float()

        if X.dim() == 1:
```

```
            X = X.reshape(1, -1)

        gaussian = GaussianModel(X, self.outcomes, self.outputs_map, self.prob_domain, self.cov)
        if self.outcomes > 1:
            gmm_weights = GaussianWeights(X, self.outcomes, self.outputs_map)
            return dist.MixtureSameFamily(gmm_weights, gaussian)
        else:
            return gaussian

    # spat and prob loss from raw outputs
    def result_metrics(self, outputs, labels):
        X = outputs.squeeze().float()
        y = labels.squeeze().float()

        if X.dim() == 1:
            X = X.reshape(1, -1)

        spat_loss = d_loss(X, y, self.outcomes, self.outputs_map).reshape(-1, 1)
        prob_loss = torch.zeros_like(spat_loss, device=X.device)
        if self.outputs_map["sigma"]:
            prob_loss = lh_loss(X, y, 1, self.single_outputs_map, self.prob_domain).reshape(-1, 1)

        return spat_loss, prob_loss

    def r2(self, outputs, labels):
        if outputs.dim() == 1:
            outputs = outputs.reshape(1, -1)

        Y = labels.repeat(1, self.outcomes) if self.outcomes > 1 else labels
        X = outputs[:, self.outputs_map["coord"][0]:self.outputs_map["coord"][1]]
        r2 = r2_score(X, Y)
        return r2

    # tensorboard metrics logging
    def log(self, writer, step, lr, train_metric, cur_batch, val_metric=None):
        def total_loss_log(metric, metric_type="total"):
            if metric_type == "val":
                atf_loss = metric[:, 0]
                folder = f"mean_val"
            else:
                atf_loss = np.mean(metric[:, :, 0], axis=0) if self.mf_handle == "mean" else
np.sum(metric[:, :, 0], axis=0)
                folder = f"current_step" if metric_type != "total" else f"mean_train"

            if self.total_loss_crit == "sum":
                total_loss = np.sum(atf_loss, axis=0)
            elif self.total_loss_crit == "mean":
                total_loss = np.mean(atf_loss, axis=0)
            elif self.total_loss_crit == "type":
                total_loss = atf_loss[1 if self.outputs_map["sigma"] else 0]

            log = f"\tTotal loss of all features:\t{total_loss}"
            print(log)
            writer.add_scalar(f"{folder}/total_loss", total_loss, step)

            if metric_type == "total":
                self.mean_epoch_train_loss = total_loss

        def spat_loss_log(metric, metric_type="total"):
            if metric_type == "val":
                spat_loss, r2 = metric[0, 0], metric[0, 1]
                folder, log = f"mean_val", f"\tGeospatial {spatial} loss:\t{spat_loss}\tCoord R2:\t{r2}"
            else:
                spat_loss, r2 = np.mean(metric[:, 0, 0], axis=0) if self.mf_handle == "mean" else
np.sum(metric[:, 0, 0], axis=0), None
                folder, log = f"current_step", f"\tGeospatial {self.mf_handle} {spatial}
loss:\t{spat_loss}"
                if metric_type == "total":
                    r2 = metric[0, 0, 1]
                    folder = f"mean_train"
                    log += f"\tCoord R2:\t{r2}"

                if metric.shape[0] > 1:
                    key_spat, minor_spat = metric[0, 0, 0], np.mean(metric[1:, 0, 0], axis=0)
                    log += f"\n\t\tKey:\t{key_spat}\tMinor:\t{minor_spat}"
                    writer.add_scalar(f"{folder}/spat_key", key_spat, step)
                    writer.add_scalar(f"{folder}/spat_minor", minor_spat, step)
```

```
            print(log)
            writer.add_scalar(f"{folder}/loss_spat", spat_loss, step)
            if r2:
                writer.add_scalar(f"{folder}/r2", r2, step)

        def prob_loss_log(metric, metric_type="total"):
            if metric_type == "val":
                prob_loss, pdf = metric[1, 0], metric[1, 1]
                folder, log = f"mean_val", f"\tProbabilistic {self.mf_handle} -LLH
loss:\t{prob_loss}\tPDF:\t{pdf}"
            else:
                prob_loss, pdf = np.mean(metric[:, 1, 0], axis=0) if self.mf_handle == "mean" else
np.sum(metric[:, 1, 0], axis=0), None
                folder, log = f"current_step", f"\tProbabilistic {self.mf_handle} -LLH loss:\t{prob_loss}"
                if metric_type == "total":
                    pdf = metric[0, 1, 1]
                    folder = f"mean_train"
                    log += f"\tPDF:\t{pdf}"

                if metric.shape[0] > 1:
                    key_prob, minor_prob = metric[0, 1, 0], np.mean(metric[1:, 1, 0], axis=0)
                    log += f"\n\t\tKey:\t{key_prob}\tMinor:\t{minor_prob}"
                    writer.add_scalar(f"{folder}/prob_key", key_prob, step)
                    writer.add_scalar(f"{folder}/prob_minor", minor_prob, step)

            print(log)
            writer.add_scalar(f"{folder}/loss_prob", prob_loss, step)
            if pdf:
                writer.add_scalar(f"{folder}/pdf", pdf, step)

        spatial = 'D^2' if self.dist else 'Coord'
        processed_metric = train_metric[0:cur_batch, :]
        mean_metric = np.mean(processed_metric, axis=0)
        current_batch_metric = processed_metric[-1, :]
        # print(current_batch_metric)

        print(f"LOG\tCurrent step: {step}\tLR:\t{lr}")
        total_loss_log(current_batch_metric, "current")
        spat_loss_log(current_batch_metric, "current")
        if self.outputs_map["sigma"]:
            prob_loss_log(current_batch_metric, "current")

        print(f"LOG\tTRAIN\tMean metrics:")
        total_loss_log(mean_metric, "total")
        spat_loss_log(mean_metric, "total")
        if self.outputs_map["sigma"]:
            prob_loss_log(mean_metric, "total")

        if val_metric is not None:
            mean_val_metric = np.mean(val_metric, axis=0)

            print(f"LOG\tVAL\tMean metrics:")
            total_loss_log(mean_val_metric, "val")
            spat_loss_log(mean_val_metric, "val")
            if self.outputs_map["sigma"]:
                prob_loss_log(mean_val_metric, "val")

        writer.flush()
```

Файл result_manager.py

```
import pandas as pd

from geopy import distance
import geopy.distance

from datetime import datetime

import scipy.sparse as sparse
from scipy.ndimage.filters import maximum_filter

from utils.benchmarks import *

# evaluation results manager
```

```python
# GM/GMM
def GaussianModel(means, sigma):
    return dist.MultivariateNormal(torch.from_numpy(means), torch.from_numpy(sigma))


# GMM weights
def GaussianWeights(weights):
    return dist.Categorical(torch.from_numpy(weights))


# GMM complete
def get_gm_family(outcomes, means, sigma, weights):
    means, sigma = means.reshape(outcomes, 2), sigma.reshape(outcomes, 2, 2)
    gaussian = GaussianModel(means, sigma)
    if weights is not None:
        gmm_weights = GaussianWeights(weights.reshape(-1))
        gm = dist.MixtureSameFamily(gmm_weights, gaussian)
    else:
        gm = gaussian
    return gm


# generating map grid with intergrid peaks
def map_grid(peaks, step=10):
    xmin, xmax = -180, 180
    ymin, ymax = -90, 90
    x = np.linspace(xmin, xmax, step)
    y = np.linspace(ymin, ymax, step)
    x = np.concatenate((x, peaks[:, 0]), axis=0)
    x = np.sort(x)
    y = np.concatenate((y, peaks[:, 1]), axis=0)
    y = np.sort(y)
    xx, yy = np.meshgrid(x, y)
    return xx, yy


# Haversine distance to the genuine truth in km for PRA metrics (outliers set to map borders)
def metric_distance(true, points, size):
    D = np.array([], dtype=float)
    longs = points[:, 0]
    lats = points[:, 1]
    longs[longs > 180] = 180
    longs[longs < -180] = -180
    lats[lats > 90] = 90
    lats[lats < -90] = -90
    for i in range(size):
        d = geopy.distance.distance((true[i, 1], true[i, 0]),
                                    (lats[i], longs[i])).km
        D = np.append(D, d)
    return D


# spatial metrics: Average/Median Distance Error (AED, MED), MSE, MAE, Acc@161
def geospatial_performance(true, dists, means, weights, outcomes, size, best, threshold=100):
    print(f"VAL\tCalculating spatial metrics for {size} samples")
    if best and outcomes > 1:
        dists, means = dists[:, 0], means[:, 0]
    else:
        if outcomes > 1:
            dists = np.sum(dists * weights, axis=1)
            mse, mae = np.zeros((size, outcomes)), np.zeros((size, outcomes))

    aed, med = np.mean(dists), np.median(dists)
    acc = (dists < threshold).sum() / size * 100
    acc161 = (dists < 161).sum() / size * 100

    if best or outcomes == 1:
        mse = np.mean((true - means)**2)
        mae = np.sum(np.abs(true - means), axis=1).mean()
    elif not best and outcomes > 1:
        for k in range(outcomes):
            mse[:, k] = np.mean(np.power(true - means[:, k], 2), axis=1)
            mae[:, k] = np.mean(np.abs(true - means[:, k]), axis=1)
```

```
        mse = np.sum(mse * weights, axis=1).mean()
        mae = np.sum(mae * weights, axis=1).mean()

    return (aed, med, acc, acc161, mse, mae)


# probabilistic metrics: Average/Median Comprehensive Accuracy Error (ACAE, MCAE), Average/Median 95%
Prediction Region Area (APRA, MPRA), 95%Coverage (COV)
def probabilistic_performance(trues, covs, means, weights, outcomes, size, best, n=100):
    print(f"VAL\tCalculating probabilistic metrics for {size} samples")
    if best and outcomes > 1:
        covs, means = covs[:, 0], means[:, 0]

    cae, pra, cov = np.zeros((size, outcomes)), \
                    np.zeros((size, outcomes)), \
                    np.zeros((size, outcomes))

    crit_chi = 5.991  # 0.95 2

    rng = np.random.default_rng()
    if not best and outcomes > 1:
        for i in range(size):
            for k in range(outcomes):
                mean, covar, true, weight = means[i, k], covs[i, k], trues[i], weights[i, k]
                gaus_sample = rng.multivariate_normal(mean, covar, n)
                rep_true = np.repeat(true.reshape(1, 2), n, axis=0).reshape(n, 2)
                gaus_dist = metric_distance(rep_true, gaus_sample, n)

                cae[i, k] = np.mean(gaus_dist, axis=0) * weight

                sigma = np.sqrt(covar[0, 0])
                error = np.sqrt(np.sum((rep_true - mean)**2))
                pra[i, k] = np.pi * sigma * crit_chi * weight
                cov[i, k] = 1 if error/sigma <= crit_chi else 0

        pra, cae = np.sum(pra, axis=1), np.sum(cae, axis=1)

    else:
        for i in range(size):
            mean, covar, true = means[i], covs[i], trues[i]

            gaus_sample = rng.multivariate_normal(mean, covar, n)
            rep_true = np.repeat(true.reshape(1, 2), n, axis=0).reshape(n, 2)
            gaus_dist = metric_distance(rep_true, gaus_sample, n)

            cae[i] = np.mean(gaus_dist, axis=0)

            sigma = covar[0, 0]
            error = np.sum((rep_true - mean)**2)

            pra[i] = np.pi * sigma * crit_chi
            cov[i] = 1 if error/sigma <= crit_chi else 0

    acae, mcae, apra, mpra, cov = np.mean(cae), np.median(cae), \
                                  np.mean(pra), np.median(pra), \
                                  cov.mean()

    return (acae, mcae, apra, mpra, cov)


class ResultManager():
    def __init__(self, val_df, text, feature, device, model_benchmark, scaled, by_user=False,
prefix=None):
        self.cluster = device.type == "cuda"
        self.feature = feature
        if prefix is None:
            prefix = feature
        self.prefix = prefix

        self.model_bm = model_benchmark

        self.scaled = scaled
        self.by_user = by_user

        self.outcomes = self.model_bm.outcomes
        self.cov = self.model_bm.cov
        self.weighted = self.model_bm.weighted
```

```
        self.dist = self.model_bm.dist

        self.outputs_map = self.model_bm.outputs_map

        self.prob = self.cov is not None

        self.covariances = {'spher': 1,
                            'diag': 2,
                            'tied': 3,
                            'full': 3}

        self.pred_columns = {}
        for i in range(self.outcomes):
            self.pred_columns[f"O{i+1}_point"] = "str"
            self.pred_columns[f"O{i+1}_dist"] = "float"
            if self.outcomes > 1:
                    self.pred_columns[f"O{i+1}_weight"] = "float"
            if self.prob:
                self.pred_columns[f"O{i+1}_sigma"] = "str"

        self.text = text
        if self.text is None:
            print(f"RESULT\tInitializing dataframe with {len(self.pred_columns)} columns for
{self.outcomes} outcome(s)")
            print(f"column tag:\ttype")
            for key, value in self.pred_columns.items():
                print(f"{key}:\t{value}")

        if val_df is not None:
            val_df.reset_index(drop=True, inplace=True)
            self.df = val_df
            self.true = val_df[["lon", "lat"]].to_numpy()
            self.size = len(self.df.index)
            if self.by_user:
                self.users = len(self.df['USER-ONLY'].unique())
        else:
            self.df = pd.DataFrame()
            self.size = 1
            self.true = np.array([])

        self.means = None
        self.dists = None
        self.weights = None
        self.covs = None

    # load evaluation results df to class entities
    def load_df(self, filename, sorting=True):
        print(f"VAL\tLOAD\tLoading dataset from {filename}")
        self.df = pd.read_json(path_or_buf=filename, lines=True)
        self.size = len(self.df.index)
        if self.by_user:
            self.users = len(self.df['USER-ONLY'].unique())
        self.true = self.df[["lon", "lat"]].to_numpy()

        if self.outcomes > 1:
            means = np.empty((0, self.outcomes, 2), float)
            dists = np.empty((0, self.outcomes))
            weights = np.empty((0, self.outcomes))
            if self.prob:
                covs = np.empty((0, self.outcomes, 2, 2), float)
        else:
            means = np.empty((0, 2), float)
            dists = np.empty((0, 1), float)
            if self.prob:
                covs = np.empty((0, 2, 2), float)

        for i in range(self.size):
            if self.outcomes > 1:
                mean_row = np.empty((0, 2), float)
                dists_row = np.empty((0, 1), float)
                weight_row = np.empty((0, 1), float)
                if self.prob:
                    cov_row = np.empty((0, 2, 2), float)

                for o in range(self.outcomes):
                    mean_row = np.append(mean_row, np.array(self.df.loc[i, f"O{o+1}_point"]).reshape(-1,
2), axis=0)
```

```
                    dists_row = np.append(dists_row, np.array(self.df.loc[i, f"O{o+1}_dist"]).reshape(-1,
1), axis=0)
                    weight_row = np.append(weight_row, self.df.loc[i, f"O{o+1}_weight"].reshape(-1, 1),
axis=0)
                    if self.prob:
                        cov_row = np.append(cov_row, np.array(self.df.loc[i, f"O{o+1}_sigma"]).reshape(-1,
2, 2), axis=0)

                means = np.append(means, mean_row.reshape(-1, self.outcomes, 2), axis=0)
                dists = np.append(dists, dists_row.reshape(-1, self.outcomes), axis=0)
                weights = np.append(weights, weight_row.reshape(-1, self.outcomes), axis=0)
                if self.prob:
                    covs = np.append(covs, cov_row.reshape(-1, self.outcomes, 2, 2), axis=0)
            else:
                means = np.append(means, np.array(self.df.loc[i, "O1_point"]).reshape(-1, 2), axis=0)
                dists = np.append(dists, np.array(self.df.loc[i, "O1_dist"]).reshape(-1, 1), axis=0)
                if self.prob:
                    covs = np.append(covs, np.array(self.df.loc[i, "O1_sigma"]).reshape(-1, 2, 2), axis=0)

        self.means = means
        self.dists = dists
        self.weights = weights if self.outcomes > 1 else None
        self.covs = covs if self.prob else None

        print(f"VAL\tLOAD\tDataset of {self.size} samples is loaded")
        if sorting and self.outcomes > 1:
            self.sort_outcomes()

        if self.outcomes == 1:
            self.df["O1_point"] = sparse.coo_matrix(self.means, shape=(self.size, 2)).toarray().tolist()
            self.df["O1_dist"] = self.dists
            if self.prob:
                self.df["O1_sigma"] = sparse.coo_matrix(self.covs.reshape((self.size, 4)),
shape=(self.size, 4)).toarray().tolist()
        else:
            for i in range(self.outcomes):
                self.df[f"O{i+1}_point"] = sparse.coo_matrix(self.means[:, i, :], shape=(self.size,
2)).toarray().tolist()
                self.df[f"O{i+1}_dist"] = self.dists[:, i]
                if self.prob:
                    self.df[f"O{i+1}_sigma"] = sparse.coo_matrix(self.covs[:, i, :].reshape((self.size,
4)), shape=(self.size, 4)).toarray().tolist()
                    self.df[f"O{i+1}_weight"] = self.weights[:, i]

    def save_df(self, prefix=None, filename=None):
        if prefix is None:
            prefix = self.prefix
        if filename is None:
            filename = f"results/val-data/{prefix}_predicted_N{self.size}_{'U' if self.by_user else ''}VF-
{self.feature}_{datetime.today().strftime('%Y-%m-%d')}.jsonl"

        with open(filename, "w") as f:
            self.df.to_json(f, orient='records', lines=True)
        print(f"VAL\tSAVE\tPredicted data of {self.size} samples is written to file: {filename}")

    # add loss metrics to df
    def metrics(self, val_metric):
        if self.prob:
            metrics_df = pd.DataFrame(
                {f'{"dist" if self.dist else "coord"}_loss': val_metric[:, 0, 0].reshape(-1),
                 f'lh_loss': val_metric[:, 1, 0].reshape(-1),
                 'pdf': val_metric[:, 1, 1].reshape(-1)})
        else:
            metrics_df = pd.DataFrame({f'{"dist" if self.dist else "coord"}_loss': val_metric[:, 0,
0].reshape(-1)})

        print(f"RESULT\tAdding metrics column(s) {', '.join(str(col) for col in
metrics_df.columns.values.tolist())} to dataframe")
        self.df = pd.concat([self.df, metrics_df], axis=1)

    # results spatial metrics
    def spatial_metric(self, threshold=100, best=True):
        if self.by_user and self.prob:
            self.users_summary()
            aed, med, acc, acc161, mse, mae = geospatial_performance(self.user_true,
                                                                     self.user_dists,
                                                                     self.user_means,
```

```
                                                                None,
                                                                1,
                                                                self.users,
                                                                best, threshold)
        else:
            aed, med, acc, acc161, mse, mae = geospatial_performance(self.true,
                                                                self.dists,
                                                                self.means,
                                                                self.weights,
                                                                self.outcomes,
                                                                self.size,
                                                                best, threshold)

        loss_dist = np.array(self.df["dist_loss"])
        loss_dist = loss_dist * 10000 if self.scaled else loss_dist

        print(f"VAL\tSpatial metrics {'best outcome' if best else ''}{'weighted outcomes' if self.outcomes
>1 and not best else ''}:"
              f"\n\tAED: {round(aed, 2)} km\t- avg error\n\tMED: {round(med, 2)} km\t- median error"
              f"\n\tMSE: {round(mse, 2)}\t- mean error^2 (degrees)\n\tMAE: {round(mae, 2)}\t- mean abs
error (degrees)"
              f"\n\tLoss D^2:\t{round(loss_dist.mean(), 2)}\t- avg loss degrees")
        print(f"\tAccuracy (<100km): {round(acc, 2)}%\t- below threshold"
              f"\n\tAccuracy (<161km): {round(acc161, 2)}%\t- below threshold")

        return aed, med, mse, mae, acc, acc161

    # results probabilistic metrics
    def prob_metric(self, best=True, n=100):
        acae, mcae, apra, mpra, cov = probabilistic_performance(self.true,
                                                                self.covs,
                                                                self.means,
                                                                self.weights,
                                                                self.outcomes,
                                                                self.size,
                                                                best, n)

        loss_llh = np.array(self.df["lh_loss"]).mean()
        pdf = np.array(self.df["pdf"]).mean()

        print(f"VAL\tGMM metrics {'best outcome' if best else ''}{'weighted outcomes' if self.outcomes > 1
and not best else ''}:"
              f"\n\tACAE: {round(acae, 2)} km\t- avg GMM sample error\n\tMCAE: {round(mcae, 2)} km\t-
median GMM sample error"
              f"\n\tAPRA (0.95): {round(apra, 2)} km2\t- avg GMM area\n\tMPRA (0.95): {round(mpra, 2)}
km2\t- median GMM area"
              f"\n\tNLLH loss:\t{round(loss_llh, 2)}\t- avg GMM neg log-likelihood loss"
              f"\n\tPDF:\t{round(pdf*100, 2)}%\t- avg GMM fit likelihood"
              f"\n\tCOV (0.95): {round(cov*100, 2)}%\t- GMM coverage")

        return acae, mcae, apra, mpra, cov

    # all results metrics
    def result_metrics(self, best=True, threshold=100):
        print(f"VAL\tCalculating spatial {'and probabilistic ' if self.prob else ''}metrics "
              f"for {self.size} result samples {'per user' if self.by_user else 'per tweet'}")

        aed, med, mse, mae, acc, acc161 = self.spatial_metric(threshold, best)
        spat_metric = [["Average SAE", aed],
                       ["Median SAE", med],
                       ["MSE", mse],
                       ["MAE", mae],
                       [f"Acc@{threshold}", acc],
                       ["Acc@161", acc161]]

        if self.prob and not self.by_user:
            acae, mcae, apra, mpra, cov = self.prob_metric(best)
            prob_metric = [["Average CAE", acae],
                           ["Median CAE", mcae],
                           ["Average 95% PRA", apra],
                           ["Median 95% PRA", mpra],
                           ["PRA COVerage", cov]]
        else:
            prob_metric = []

        out = "BEST" if best else f"ALL {self.outcomes}"
        return [["Outcome", out]] + spat_metric + prob_metric
```

```
    # full performance measure of the model
    def performance(self, save=True):
        best_metric = self.result_metrics(True)
        self.performance_df = pd.DataFrame(best_metric, columns=["metric", "value"])

        if self.outcomes > 1 and not self.by_user:
            all_metric = self.result_metrics(False)
            self.performance_df = self.performance_df.append(pd.DataFrame(all_metric, columns=["metric",
"value"]), ignore_index=True)

        self.performance_df['metric'] = self.performance_df['metric'].apply(lambda x: "{:<20}".format(x))

        if save:
            filename = f"results/metric/{self.prefix}_metric_N{self.users if self.by_user else
self.size}_" \
                       f"{'U' if self.by_user else ''}VF-{self.feature}_{datetime.today().strftime('%Y-%m-
%d')}.txt"

            with open(filename, "w") as f:
                self.performance_df.to_csv(f, header=False, index=False, sep="\t", mode="a")
            print(f"VAL\tSAVE\tPerformance metrics of {self.users if self.by_user else self.size} samples
are written to file: {filename}")

    # per user summary
    def users_summary(self):
        print(f"VAL\tSummarizing predictions for {self.users} users")
        self.user_true, self.user_means = np.zeros((self.users, 2), float), \
                                          np.zeros((self.users, 2), float)
        users = self.df['USER-ONLY'].unique()
        for u in range(self.users):
            ids = self.df.index[self.df['USER-ONLY'] == users[u]].tolist()

            user_true = self.true[ids]
            values, counts = np.unique(user_true, return_counts=True, axis=0)
            if any(i > 1 for i in counts):
                user_true = values[np.argmax(counts)]
            else:
                time = self.df.loc[self.df.index[ids], 'date']
                user_true = self.df.loc[self.df.index[time.idxmin()], ['lon', "lat"]].to_numpy()
            self.user_true[u] = user_true

            user_samples = len(ids)
            # print(user_samples, users[u])
            if user_samples > 1000:
                # print("skip")
                continue
            user_peaks = self.means[ids].reshape(-1, 2) if self.outcomes == 1 else self.means[ids, 0,
:].reshape(-1, 2)
            peaks_unique, ind = np.unique(np.round(user_peaks, 4), return_index=True, axis=0)
            xx, yy = map_grid(user_peaks[ind], 10)
            XX = np.array([xx.ravel(), yy.ravel()]).T

            Zexp = np.zeros_like(xx).flatten()
            user_means, user_covs, user_weights = self.means[ids], self.covs[ids], self.weights[ids]
            for i in range(user_samples):
                i_weights = user_weights[i, :] if self.outcomes > 1 else None
                gmm = get_gm_family(self.outcomes, user_means[i, :], user_covs[i, :], i_weights)
                Zexp += np.exp(gmm.log_prob(torch.from_numpy(XX)).numpy())

            Z = Zexp / user_samples
            Z = Z.reshape(xx.shape)
            local_max_indexes = np.where(1 == (Z == maximum_filter(Z, mode="nearest", size=(10, 10))))
            ind = np.ravel_multi_index(local_max_indexes, Z.shape)

            max_Z, max_XX = Zexp[ind], XX[ind]
            max_XX_uni, ind_uni = np.unique(max_XX, return_index=True, axis=0)
            max_Z_uni = max_Z[ind_uni]

            top = 5 if len(ind_uni) > 5 else len(ind_uni)
            ind_top_5 = (-max_Z_uni).argsort()[:top]
            peaks = max_XX_uni[ind_top_5]

            self.user_means[u] = peaks[0]

        self.user_dists = metric_distance(self.user_true, self.user_means, self.users).reshape(self.users,
1)
```

```
    # sort per-tweet outcomes by their weights
    def sort_outcomes(self):
        if self.dists is None and self.true.size > 0:
            self.dists = self.distances(self.means)

        print(f"RESULT\tSorting all outputs for {self.outcomes} outcomes by probabilistic weights")

        if self.text:
            sort_indexes = np.argsort(self.weights[0])
            index = sort_indexes[::-1]
            self.means[0] = self.means[0, index]
            if self.outcomes > 1:
                self.weights[0] = self.weights[0, index]
            if self.prob:
                self.covs[0] = self.covs[0, index]

        else:
            sort_indexes = np.argsort(self.weights, axis=1)

            for i in range(self.size):
                index = sort_indexes[i][::-1]
                self.means[i, :] = self.means[i, index]
                self.weights[i, :] = self.weights[i, index]
                if self.dists is not None:
                    self.dists[i, :] = self.dists[i, index]
                if self.prob:
                    self.covs[i, :] = self.covs[i, index]

    # add to df plain spat outputs
    def coord_outputs(self, predicted):
        self.means = np.multiply(predicted[:, self.outputs_map["coord"][0]:self.outputs_map["coord"][1]],
100) if self.scaled else predicted[:, self.outputs_map["coord"][0]:self.outputs_map["coord"][1]]
        self.means = self.means.reshape(self.size, 2) if self.outcomes == 1 else
self.means.reshape(self.size, self.outcomes, 2)

        if self.outputs_map["weight"]:
            self.weights = weights(torch.from_numpy(predicted), self.outcomes, self.outputs_map).numpy()
        else:
            self.weights = None

        if self.true.size > 0:
            self.dists = self.distances(self.means)

        if self.outcomes > 1:
            #print(f"RESULT\tSorting all outputs for {self.outcomes} outcomes by distances error")
            self.sort_outcomes()

        if self.df.size > 0:
            spat_df = pd.DataFrame({column: pd.Series(dtype=type) for column, type in
self.pred_columns.items()})

            if self.outcomes == 1:
                spat_df["O1_point"] = sparse.coo_matrix(self.means, shape=(self.size,
2)).toarray().tolist()
                spat_df["O1_dist"] = self.dists
            else:
                for i in range(self.outcomes):
                    spat_df[f"O{i+1}_weight"] = self.weights[:, i]
                    spat_df[f"O{i+1}_point"] = sparse.coo_matrix(self.means[:, i, :], shape=(self.size,
2)).toarray().tolist()
                    spat_df[f"O{i+1}_dist"] = self.dists[:, i]

            print(f"RESULT\tSetting spatial output columns {', '.join(str(col) for col in
spat_df.columns.values.tolist())} in dataframe")
            self.df = pd.concat([self.df, spat_df], axis=1)

    # add to df prob outputs (reading GM/GMM models)
    def soft_outputs(self, pm):
        self.prob_models = pm

        if self.outcomes > 1:
            means = np.empty((0, self.outcomes, 2), float)
            covs = np.empty((0, self.outcomes, 2, 2), float)
            weights = np.empty((0, self.outcomes))
        else:
            means = np.empty((0, 2), float)
```

```
            covs = np.empty((0, 2, 2), float)

        for models in self.prob_models:
            if self.outcomes > 1:
                if self.cluster:
                    means = np.append(means, models.component_distribution.loc.cpu().numpy(), axis=0)
                    covs = np.append(covs, models.component_distribution.covariance_matrix.cpu().numpy(),
axis=0)
                    weights = np.append(weights, models.mixture_distribution.probs.cpu().numpy(), axis=0)
                else:
                    means = np.append(means, models.component_distribution.loc.numpy(), axis=0)
                    covs = np.append(covs, models.component_distribution.covariance_matrix.numpy(),
axis=0)
                    weights = np.append(weights, models.mixture_distribution.probs.numpy(), axis=0)
            else:
                if self.cluster:
                    means = np.append(means, models.loc.cpu().numpy().reshape(-1, 2), axis=0)
                    covs = np.append(covs, models.covariance_matrix.cpu().numpy().reshape(-1, 2, 2),
axis=0)
                else:
                    means = np.append(means, models.loc.numpy().reshape(-1, 2), axis=0)
                    covs = np.append(covs, models.covariance_matrix.numpy().reshape(-1, 2, 2), axis=0)

        self.means = np.multiply(means, 100) if self.scaled else means
        self.covs = np.multiply(covs, 100) if self.scaled else covs

        if self.true.size > 0:
            self.dists = self.distances(self.means)

        if self.outcomes > 1:
            self.weights = weights
            self.sort_outcomes()
        else:
            self.weights = None

        if self.df.size > 0:
            prob_models_df = pd.DataFrame({column: pd.Series(dtype=type) for column, type in
self.pred_columns.items()})

            if self.outcomes == 1:
                prob_models_df["O1_point"] = sparse.coo_matrix(self.means, shape=(self.size,
2)).toarray().tolist()
                prob_models_df["O1_sigma"] = sparse.coo_matrix(self.covs.reshape((self.size, 4)),
shape=(self.size, 4)).toarray().tolist()
                prob_models_df["O1_dist"] = self.dists
            else:
                for i in range(self.outcomes):
                    prob_models_df[f"O{i+1}_point"] = sparse.coo_matrix(self.means[:, i, :],
shape=(self.size, 2)).toarray().tolist()
                    prob_models_df[f"O{i+1}_sigma"] = sparse.coo_matrix(self.covs[:, i,
:].reshape((self.size, 4)), shape=(self.size, 4)).toarray().tolist()
                    prob_models_df[f"O{i+1}_weight"] = self.weights[:, i]
                    prob_models_df[f"O{i+1}_dist"] = self.dists[:, i]

            print(f"RESULT\tSetting spatial and probabilistic output columns {', '.join(str(col) for col
in prob_models_df.columns.values.tolist())} in dataframe")
            self.df = pd.concat([self.df, prob_models_df], axis=1)

    # Haversine distances to the genuine truth in km for coords
    def distances(self, points, true=None, size=None):
        if true is None:
            true = self.true
        if size is None:
            size = self.size

        print(f"RESULT\tCalculating distances of {size} samples with {self.outcomes} outcome(s)")
        if self.outcomes == 1:
            D = np.array([], dtype=float)
            for i in range(size):
                d = geopy.distance.distance((true[i, 1], true[i, 0]),
                                            (points[i, 1], points[i, 0])).km
                D = np.append(D, d)
        else:
            D = np.empty((0, self.outcomes), dtype=float)
            for i in range(size):
                row = np.array([])
                for j in range(self.outcomes):
```

```
                    d = geopy.distance.distance((true[i, 1], true[i, 0]),
                                                (points[i, j, 1], points[i, j, 0])).km
                    row = np.append(row, d)
                D = np.append(D, row.reshape(1, self.outcomes), axis=0)
        return D
```

Файл result_visuals.py

```
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
from matplotlib.colors import LogNorm
from mpl_toolkits.basemap import Basemap

import seaborn as sns

import plotly.graph_objects as go
from pathlib import Path

from scipy.special import softmax

import os
import shutil

import imageio
from moviepy.editor import *

from utils.result_manager import *

# visualization of evaluation results


def plot_gmm(samples, outcomes, means, covs, weights, cluster, title, filename, save=True):
    palette = {1: 'darkgreen',
               2: 'goldenrod',
               3: 'darkorange',
               4: 'crimson',
               5: 'darkred'}

    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 15))

    xmin, xmax = -180, 180
    ymin, ymax = -90, 90
    step_big = 45.
    ticks_big_x, ticks_big_y = range(int(xmin), int(xmax), int(step_big)), \
                               range(int(ymin), int(ymax), int(step_big))
    tick_labels_big_x, tick_labels_big_y = [str(x) for x in ticks_big_x], \
                                           [str(y) for y in ticks_big_y]

    total_peaks = means.reshape(-1, 2)
    peaks_unique, ind = np.unique(np.round(total_peaks, 4), return_index=True, axis=0)
    intergrid_peaks = total_peaks[ind]
    print(f"Total number of GMM peaks {total_peaks.shape[0]} - {intergrid_peaks.shape[0]} unique to
include")
    grid_step = 50 if samples > 1 else 400
    xxb, yyb = map_grid(intergrid_peaks, grid_step)
    XX_big = np.array([xxb.ravel(), yyb.ravel()]).T

    if samples > 1:
        Z_big, Z_big_exp = np.zeros_like(xxb).flatten(), np.zeros_like(xxb).flatten()
        for i in range(samples):
            i_weights = weights[i, :] if outcomes > 1 else None
            gmm = get_gm_family(outcomes, means[i, :], covs[i, :], i_weights)
            Z_big += gmm.log_prob(torch.from_numpy(XX_big)).numpy()
            Z_big_exp += np.exp(gmm.log_prob(torch.from_numpy(XX_big)).numpy())

        Z_big_exp = Z_big_exp / samples
        Z_big = Z_big / samples

        Z = Z_big_exp.reshape(xxb.shape)
        local_max_indexes = np.where(1 == (Z == maximum_filter(Z, mode="nearest", size=(10, 10))))
        ind = np.ravel_multi_index(local_max_indexes, Z.shape)

        max_Z = Z_big_exp[ind]
        max_XX = XX_big[ind]
        max_XX_uni, ind_uni = np.unique(max_XX, return_index=True, axis=0)
        max_Z_uni = max_Z[ind_uni]
        print(f"Found {max_XX.shape[0]} local maximums - {max_XX_uni.shape[0]} unique peaks")
```

```
    top = 5 if len(ind_uni) > 5 else len(ind_uni)
    ind_top_5 = (-max_Z_uni).argsort()[:top]
    peaks = max_XX_uni[ind_top_5]
    p_weights = softmax(max_Z_uni[ind_top_5])
    for i in range(top):
        pdf = np.round(max_Z_uni[ind_top_5][i], 5)
        weight = np.round(p_weights[i] * 100, 2)
        point = f"lon: {'  lat: '.join(map(str, peaks[i])) }"
        print(f"\t{i}\t{weight}%\t-\t{point}\t-\t{pdf}")

    ind = np.argwhere(np.round(p_weights * 100, 2) > 0)
    print(f"Regressing to top {top} - {len(ind)} significant peaks")
    significant = peaks[ind].reshape(len(ind), 2)
    sig_weights = p_weights[ind].flatten()

else:
    gmm = get_gm_family(outcomes, means, covs, weights)
    Z_big = gmm.log_prob(torch.from_numpy(XX_big)).numpy()

    if outcomes > 1:
        ind = np.argwhere(np.round(weights * 100, 2) > 0)
        significant = means[ind].reshape(-1, 2)
        sig_weights = weights[ind].flatten()
    else:
        significant = means.reshape(-1, 2)
        sig_weights = np.ones(1)

for i in range(len(sig_weights)):
    weight = np.round(sig_weights[i] * 100, 2)
    point = f"lon: {'  lat: '.join(map(str, significant[i])) }"
    if weight > 0:
        print(f"\tOut {i+1}\t{weight}%\t-\t{point}")

margin_lon, margin_lat = 10, 5
min_lon, max_lon = min(significant[:, 0]) - margin_lon, max(significant[:, 0]) + margin_lon
min_lat, max_lat = min(significant[:, 1]) - margin_lat, max(significant[:, 1]) + margin_lat
step_zoom = 15.0
ticks_zoom_x, ticks_zoom_y = range(int(min_lon), int(max_lon), int(step_zoom)), \
                             range(int(min_lat), int(max_lat), int(step_zoom))
tick_labels_zoom_x, tick_labels_zoom_y = [str(x) for x in ticks_zoom_x], \
                                         [str(y) for y in ticks_zoom_y]
xxz, yyz = np.mgrid[min_lon:max_lon:400j, min_lat:max_lat:400j]
XX_zoom = np.array([xxz.ravel(), yyz.ravel()]).T

if samples > 1:
    Z_zoom = np.zeros_like(xxz).flatten()
    for i in range(samples):
        i_weights = weights[i, :] if outcomes > 1 else None
        gmm = get_gm_family(outcomes, means[i, :], covs[i, :], i_weights)
        Z_zoom += np.exp(gmm.log_prob(torch.from_numpy(XX_zoom)).numpy())

    Z_zoom = Z_zoom / samples
else:
    gmm = get_gm_family(outcomes, means, covs, weights)
    Z_zoom = np.exp(gmm.log_prob(torch.from_numpy(XX_zoom)).numpy())

Z_big, Z_zoom = Z_big.reshape(xxb.shape), Z_zoom.reshape(xxz.shape)
zbmin, zzmin = np.min(Z_big), np.min(Z_zoom)
zbmax, zzmax = np.max(Z_big), np.max(Z_zoom)

ax_big = ax[0]

ax_big.set_xlim(xmin, xmax)
ax_big.set_ylim(ymin, ymax)
ax_big.set_title(f'Log-Likelihood score of {samples} GMM{"s" if samples > 2 else "" }', y=1.0, pad=24)

map_big = Basemap(ax=ax_big, projection='mill', resolution='l')
map_big.drawcoastlines(linewidth=0.5, color="black", zorder=2)
map_big.drawcountries(linewidth=0.7, color="black", zorder=3)
map_big.drawparallels(np.arange(ymin, ymax, step_big), labels=tick_labels_big_y)
map_big.drawmeridians(np.arange(xmin, xmax, step_big), labels=tick_labels_big_x)
map_big.drawmapboundary(fill_color='lightgrey', zorder=0)
map_big.fillcontinents(color='white', lake_color='lightgrey', zorder=1)

contour_big = map_big.contourf(xxb, yyb, Z_big, levels=np.linspace(zbmin, zbmax, 250),
                               cmap='Spectral_r', alpha=0.7, zorder=9, latlon=True)
```

```
    plt.colorbar(contour_big, ax=ax_big, orientation="horizontal", pad=0.2)

    ax_zoom = ax[1]
    ax_zoom.set_xlim(min_lon, max_lon)
    ax_zoom.set_ylim(min_lat, max_lat)
    ax_zoom.set_title('Max Probability Density Function region', y=1.0, pad=24)

    map_zoom = Basemap(ax=ax_zoom, projection='mill',
                       llcrnrlat = min_lat,
                       llcrnrlon = min_lon,
                       urcrnrlat = max_lat,
                       urcrnrlon = max_lon, resolution='h')
    map_zoom.drawcoastlines(linewidth=0.5, color="black", zorder=2)
    map_zoom.drawcountries(linewidth=0.7, color="black", zorder=3)
    map_zoom.drawmapboundary(fill_color='lightgrey', zorder=0)
    map_zoom.drawparallels(np.arange(min_lat, max_lat, step_zoom), labels=tick_labels_zoom_y)
    map_zoom.drawmeridians(np.arange(min_lon, max_lon, step_zoom), labels=tick_labels_zoom_x)
    map_zoom.fillcontinents(color='white', lake_color='lightgrey', zorder=1)

    Z_zoom = np.ma.array(Z_zoom, mask=Z_zoom < 1e-8)
    contour_zoom = map_zoom.contourf(xxz, yyz, Z_zoom, levels=np.linspace(zzmin, zzmax, 250),
                                     cmap='Spectral_r', alpha=0.7, zorder=9, latlon=True)
    plt.colorbar(contour_zoom, ax=ax_zoom, orientation="horizontal", pad=0.2)

    for i in range(len(sig_weights)):
        color = palette[i+1] if i < 5 else palette[5]
        point = f"lon: {'  lat: '.join(map(str, significant[i])) }"
        label = point + f" - {round(sig_weights[i] * 100, 2)}%",
        map_zoom.scatter(significant[i, 0], significant[i, 1], latlon=True,
                         label=label, color=color,
                         s=10, zorder=9999)
        map_zoom.scatter(significant[i, 0], significant[i, 1], latlon=True,
                         color=color, alpha=0.2,
                         s=100, zorder=9999)

    plt.legend(loc='upper center', title="Predicted outcomes", bbox_to_anchor=(0.5, -0.1), fancybox=True,
shadow=True)
    plt.suptitle(title)

    if save:
        plt.savefig(filename, dpi=300)
        print(f"VAL\tSAVE\tGaussian model is drawn to file: {filename}")
    if not cluster:
        plt.show()


class ResultVisuals():
    def __init__(self, manager):
        self.manager = manager
        self.cluster = self.manager.cluster
        self.feature = self.manager.feature
        self.prefix = self.manager.prefix

        self.df = self.manager.df
        self.pred_columns = self.manager.pred_columns
        self.size = self.manager.size
        self.true = self.manager.true

        self.prob = self.manager.prob
        self.outcomes = self.manager.outcomes
        self.cov = self.manager.cov
        self.weighted = self.manager.weighted

        self.dist = self.manager.dist

        self.covariances = self.manager.covariances

        # palette for sorted by weight outcomes
        self.palette = {1: 'darkgreen',
                        2: 'goldenrod',
                        3: 'darkorange',
                        4: 'crimson',
                        5: 'darkred'}

    # multiple tweets GMM NLLH contour on the map
    def prob_map_animation(self, frames=42, gif=False, clean=True, video=True):
        frames = self.size if frames > self.size else frames
```

```
        xmin, xmax = -180, 180
        ymin, ymax = -90, 90
        xx, yy = np.mgrid[xmin:xmax:200j, ymin:ymax:200j]
        XX = np.array([xx.ravel(), yy.ravel()]).T

        map_frames_files = []
        frame_dir = f"results/img/gaussian_{self.prefix}_N{frames}_{datetime.today().strftime('%Y-%m-
%d')}"
        Path(f"./{frame_dir}").mkdir(parents=True, exist_ok=True)

        for i in range(frames):
            loss_lh = self.df.loc[i, f'lh_loss']

            fig, ax = plt.subplots(figsize=(20, 20))
            ax.set_xlim(xmin, xmax)
            ax.set_ylim(ymin, ymax)

            weights = self.manager.weights[i, :] if self.outcomes > 1 else None
            gmm = get_gm_family(self.outcomes, self.manager.means[i, :], self.manager.covs[i, :], weights)

            Z = gmm.log_prob(torch.from_numpy(XX)).numpy()
            gmm_lh = gmm.log_prob(torch.from_numpy(self.true[i, :])).numpy()

            zmin = np.min(Z)
            zmax = np.max(Z)
            print(f"{i}\tMin: {zmin}\tMax: {zmax}")
            Z = Z.reshape(xx.shape)

            #contour = ax.contour(xx, yy, Z, levels=np.linspace(zmin, zmax, 500), cmap='RdYlGn_r',
linewidths=0.5)
            contour = ax.contourf(xx, yy, Z, levels=np.linspace(zmin, zmax, 250), cmap='Spectral_r',
alpha=0.7)
            fig.colorbar(contour, orientation="horizontal", pad=0.2)

            map = Basemap(ax=ax)
            map.drawcoastlines(linewidth=0.5, color="gray")
            map.drawcountries(linewidth=0.7, color="gray")
            map.drawparallels(np.arange(ymin, ymax, 30.))
            map.drawmeridians(np.arange(xmin, xmax, 30.))

            if self.outcomes > 1:
                map.scatter(self.manager.means[i, :, 0], self.manager.means[i, :, 1], s=1, c="black",
zorder=10)
            else:
                map.scatter(self.manager.means[i, 0], self.manager.means[i, 1], s=1, c="black", zorder=10)

            map.scatter(self.true[i, 0], self.true[i, 1], color="white", s=10, zorder=11)
            plt.axvline(x=self.true[i, 0], color="white", linestyle='--', lw=3)
            plt.axhline(y=self.true[i, 1], color="white", linestyle='--', lw=3)
            fig.text(0.5, 0.04, f"{self.true[i, :]} --- LLH: {loss_lh} --- GMM: {gmm_lh}", ha='center',
va='center')

            plt.xlabel('Longitude')
            plt.ylabel('Latitude')
            ax.set_yticks(range(-90, 90, 30))
            ax.set_xticks(range(-180, 180, 30))
            plt.title(f'{self.prefix} - Log-likelihood contour of Gaussian Model with {self.outcomes}
means - frame {i}')

            if gif or video:
                filename = f"{frame_dir}/plot-{i}.png"
                map_frames_files.append(filename)
                plt.savefig(filename, dpi=100)
                print(f"VAL\tSAVE\t{i} - Gaussian model of {self.size} samples is drawn to file:
{filename}")

        if gif:
            print(f"VAL\tCreating gif animation for {frames} gaussian plots")
            gif_filename =
f"results/gif/gaussian_maps_{self.prefix}_N{frames}_{datetime.today().strftime('%Y-%m-%d')}.gif"
            with imageio.get_writer(gif_filename, mode="I") as writer:
                for f in map_frames_files:
                    image = imageio.imread(f)
                    writer.append_data(image)
            print(f"VAL\tGIF animation for {frames} gaussian plots saved to {gif_filename}")

        if video:
```

```
            mp4_filename =
f"results/mp4/gaussian_maps_{self.prefix}_N{frames}_{datetime.today().strftime('%Y-%m-%d')}.mp4"
            clip = ImageSequenceClip(map_frames_files, fps=4)
            clip.write_videofile(mp4_filename, fps=24)

        if clean:
            print(f"VAL\tRemoving {frames} gaussian plot frames")
            shutil.rmtree(f"./{frame_dir}", ignore_errors=True)

    def summarize_prediction(self, user_index=0, samples=100, save=True):
        user = self.df['USER-ONLY'].unique()[user_index]
        ids = self.df.index[self.df['USER-ONLY'] == user].tolist()
        if samples < len(ids):
            ids = ids[0:samples]
        size = len(ids)
        means, covs = self.manager.means[ids], self.manager.covs[ids]
        weights = self.manager.weights[ids] if self.outcomes > 1 else None

        title = f'{self.prefix}\nsummary of {size} tweet GMMs with {self.outcomes} means' \
                f'\nUser: {user}'
        if save:
            filename =
f"results/img/gmm_user_summary_S{size}_{self.prefix}_N{self.size}_{datetime.today().strftime('%Y-%m-
%d')}.png"
        else:
            filename = None

        print(f"User:\t{user}")
        print(f"Estimating user geolocation from {size} tweet GMM predictions")
        plot_gmm(size, self.outcomes, means, covs, weights, self.cluster, title, filename, save)

    # single tweet GMM NLLH contour on the map
    def gaus_map(self, index=42, save=True):
        title = f'{self.prefix}\nplots of tweet GMM with {self.outcomes} means - sample {index}' \
                f'\nText: {self.df.loc[index, self.feature]}'
        if save:
            filename = f"results/img/gaussian_sample_map_ID-
{index}_{self.prefix}_N{self.size}_{datetime.today().strftime('%Y-%m-%d')}.png"
        else:
            filename = None

        means, covs, weights = self.manager.means[index], self.manager.covs[index],
self.manager.weights[index]

        plot_gmm(1, self.outcomes, means, covs, weights, self.cluster, title, filename, True)

    # single text results visualization on the map
    def text_map_result(self, index=0, save=True):
        if self.prob:
            title = f'{self.prefix}\nplots of GMM with {self.outcomes} means'
            if self.manager.text:
                title += f"\nText: {self.manager.text}\n"

            if save:
                filename = f"results/img/text_map_{self.prefix}_{datetime.today().strftime('%Y-%m-
%d')}.png"
            else:
                filename = None

            means, covs = self.manager.means[0], self.manager.covs[0]
            weights = self.manager.weights[0] if self.outcomes > 1 else None
            plot_gmm(1, self.outcomes, means, covs, weights, self.cluster, title, filename, True)
        else:

            fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 15))

            xmin, xmax = -180, 180
            ymin, ymax = -90, 90
            step_big = 45.
            ticks_big_x, ticks_big_y = range(int(xmin), int(xmax), int(step_big)), \
                                       range(int(ymin), int(ymax), int(step_big))
            tick_labels_big_x, tick_labels_big_y = [str(x) for x in ticks_big_x], \
                                                   [str(y) for y in ticks_big_y]

            if self.outcomes > 1:
                ind = np.argwhere(np.round(self.manager.weights[index, :] * 100, 2) > 0)
                significant = self.manager.means[index, ind].reshape(-1, 2)
```

```
            sig_weights = self.manager.weights[index, ind].flatten()
        else:
            significant = self.manager.means.reshape(-1, 2)
            sig_weights = np.ones(1)

        for i in range(len(sig_weights)):
            weight = np.round(sig_weights[i] * 100, 2)
            point = f"lon: {'  lat: '.join(map(str, significant[i])) }"
            if weight > 0:
                print(f"\tOut {i+1}\t{weight}%\t-\t{point}")

        margin_lon, margin_lat = 10, 5
        min_lon, max_lon = min(significant[:, 0]) - margin_lon, max(significant[:, 0]) + margin_lon
        min_lat, max_lat = min(significant[:, 1]) - margin_lat, max(significant[:, 1]) + margin_lat
        step_zoom = 5.0
        ticks_zoom_x, ticks_zoom_y = range(int(min_lon), int(max_lon), int(step_zoom)), \
                                     range(int(min_lat), int(max_lat), int(step_zoom))
        tick_labels_zoom_x, tick_labels_zoom_y = [str(x) for x in ticks_zoom_x], \
                                                 [str(y) for y in ticks_zoom_y]

        ax_big = ax[0]
        ax_big.set_xlim(xmin, xmax)
        ax_big.set_ylim(ymin, ymax)

        map_big = Basemap(ax=ax_big, projection='mill', resolution='l')
        map_big.drawcoastlines(linewidth=0.5, color="black", zorder=2)
        map_big.drawcountries(linewidth=0.7, color="black", zorder=3)
        map_big.drawparallels(np.arange(ymin, ymax, step_big), labels=tick_labels_big_y)
        map_big.drawmeridians(np.arange(xmin, xmax, step_big), labels=tick_labels_big_x)
        map_big.drawmapboundary(fill_color='lightgrey', zorder=0)
        map_big.fillcontinents(color='white', lake_color='lightgrey', zorder=1)

        if self.outcomes > 1:
            for i in range(self.outcomes):
                color = self.palette[i+1] if i < 5 else self.palette[5]
                map_big.scatter(self.manager.means[index, i, 0],
                                self.manager.means[index, i, 1],
                                color=color,
                                s=10 * self.manager.weights[index, i],
                                latlon=True,
                                zorder=9999)
                map_big.scatter(self.manager.means[index, i, 0],
                                self.manager.means[index, i, 1],
                                color=color,
                                alpha=0.2,
                                s=max(100, 1000 * self.manager.weights[index, i]),
                                latlon=True,
                                zorder=9999)
        else:
            map_big.scatter(self.manager.means[index, 0],
                            self.manager.means[index, 1],
                            latlon=True,
                            s=10, c="black", zorder=9999)
            map_big.scatter(self.manager.means[index, 0],
                            self.manager.means[index, 1],
                            latlon=True,
                            s=100, c="black", alpha=0.2, zorder=9999)

        ax_zoom = ax[1]
        ax_zoom.set_xlim(min_lon, max_lon)
        ax_zoom.set_ylim(min_lat, max_lat)

        map_zoom = Basemap(ax=ax_zoom, projection='mill',
                           llcrnrlat = min_lat,
                           llcrnrlon = min_lon,
                           urcrnrlat = max_lat,
                           urcrnrlon = max_lon, resolution='h')
        map_zoom.drawcoastlines(linewidth=0.5, color="black", zorder=2)
        map_zoom.drawcountries(linewidth=0.7, color="black", zorder=3)
        map_zoom.drawmapboundary(fill_color='lightgrey', zorder=0)
        map_zoom.drawparallels(np.arange(min_lat, max_lat, step_zoom), labels=tick_labels_zoom_y)
        map_zoom.drawmeridians(np.arange(min_lon, max_lon, step_zoom), labels=tick_labels_zoom_x)
        map_zoom.fillcontinents(color='white', lake_color='lightgrey', zorder=1)

        if self.outcomes > 1:
            for i in range(self.outcomes):
                if np.round(self.manager.weights[index, i] * 100, 2) > 0:
```

```
                        color = self.palette[i+1] if i < 5 else self.palette[5]
                        map_zoom.scatter(self.manager.means[index, i, 0],
                                         self.manager.means[index, i, 1],
                                         latlon=True,
                                         label=f"Out {i+1}: { ', '.join(map(str, self.manager.means[index,
i]))} - {round(self.manager.weights[index, i] * 100, 2)}%",
                                         color=color,
                                         s=max(1, 10 * self.manager.weights[index, i]),
                                         zorder=9999)
                        map_zoom.scatter(self.manager.means[index, i, 0],
                                         self.manager.means[index, i, 1],
                                         latlon=True,
                                         color=color,
                                         s=max(100, 1000 * self.manager.weights[index, i]),
                                         alpha=0.2,
                                         zorder=9999)
            else:
                map_zoom.scatter(self.manager.means[index, 0],
                                 self.manager.means[index, 1],
                                 latlon=True,
                                 s=10, c="black", zorder=9999)
                map_zoom.scatter(self.manager.means[index, 0],
                                 self.manager.means[index, 1],
                                 latlon=True,
                                 label=f"Out 1: { ', '.join(map(str, self.manager.means))}",
                                 s=100, c="black", alpha=0.2, zorder=9999)

            title = f'{self.prefix}\nscatter plots of {self.outcomes} points'
            if self.manager.text:
                title += f"\nText: {self.manager.text}"

            plt.legend(loc='upper center', title="Predicted outcomes", bbox_to_anchor=(0.5, -0.1),
fancybox=True, shadow=True)
            plt.suptitle(title)

            if save:
                filename = f"results/img/text_map_{self.prefix}_{datetime.today().strftime('%Y-%m-
%d')}.png"
                plt.savefig(filename, dpi=300)
                print(f"VAL\tSAVE\tScatter plot of text prediction is drawn to file: {filename}")
            if not self.cluster:
                plt.show()

    # Densities on log scale per outcome
    def density(self, save=True):
        fig, ax = plt.subplots(figsize=(20, 10))
        ax.set(xscale="log")
        ax.set_xlim(1e-1, 4e+4)

        for i in range(self.outcomes):
            label = f"{i+1}: d = {round(self.df[f'O{i+1}_dist'].median())} km; "
            if self.weighted:
                label += f" w = {round(self.df[f'O{i+1}_weight'].median(), 5)};"

            color = self.palette[i+1] if i < 5 else self.palette[5]

            sns.kdeplot(self.df[f'O{i+1}_dist'], ax=ax, label=label, bw_adjust=.5,
                        color=color, linewidth=2, fill=True, alpha=0.1)

        box = ax.get_position()
        ax.set_position([box.x0, box.y0 + box.height * 0.1, box.width, box.height * 0.9])
        ax.legend(loc='upper center', title="Median per outcome", bbox_to_anchor=(0.5, -0.1),
fancybox=True, shadow=True, ncol=2)
        plt.title(f'{self.prefix}\nDensity Plot for Distance error')
        plt.xlabel('Error (km)')
        plt.ylabel('Density')

        if save:
            filename = f"results/img/density_{self.prefix}_N{self.size}_{datetime.today().strftime('%Y-%m-
%d')}.png"
            plt.savefig(filename, dpi=300)
            print(f"VAL\tSAVE\tDistance density of {self.size} samples is drawn to file: {filename}")
        if not self.cluster:
            plt.show()

    # Cumulative Distribution per outcome
    def cum_dist(self, best=True, threshold=200, save=True):
```

```
        fig, ax = plt.subplots(figsize=(20, 10))
        ax.set(xscale="log")
        ax.set_xlim(1e-1, 4e+4)

        best_outcome_dist = "O1_dist"

        if self.outcomes == 1 or best:
            sns.ecdfplot(self.df, x=best_outcome_dist, ax=ax)

            x, y = ax.get_lines()[0].get_data()
            segments = np.array([x[:-1], y[:-1], x[1:], y[1:]]).T.reshape(-1, 2, 2)
            norm = LogNorm(self.df[best_outcome_dist].min(), self.df[best_outcome_dist].max())
            lc = LineCollection(segments, cmap='RdYlGn_r', norm=norm)
            lc.set_array(x[:-1])
            lc.set_linewidth(2)
            ax.get_lines()[0].remove()
            line = ax.add_collection(lc)
            fig.colorbar(line, ax=ax)

            plt.axvline(threshold, color="red", linestyle='dashed', lw="1")
            prop = self.df[best_outcome_dist][self.df[best_outcome_dist] < threshold].count() / self.size
            plt.axhline(prop, color="black")
            plt.text(1, prop + 0.005, f"{round(prop * 100, 2)}%", color="black", fontweight="bold")

            ax.fill_between(x, prop, y, where=x >= threshold, color='red', alpha=0.1, hatch='xx')
            ax.fill_between(x, y, where=y <= prop, color='green', alpha=0.1, hatch='xx')
            ax.fill_between(x, 0, prop, where=y >= prop, color='green', alpha=0.1, hatch='xx')

        else:
            for i in range(self.outcomes):
                label = f"{i+1}: distance = {round(self.df[f'O{i+1}_dist'].mean())} km; "
                if self.weighted:
                    label += f" weight = {round(self.df[f'O{i+1}_weight'].mean(), 5)};"
                color = self.palette[i+1] if i < 5 else self.palette[5]

                sns.ecdfplot(self.df, x=f'O{i+1}_dist', ax=ax, label=label, color=color, lw=2)

                prop = self.df[f'O{i+1}_dist'][self.df[f'O{i+1}_dist'] < threshold].count() / self.size
                plt.axhline(prop, color=color, lw=1)
                if prop > 0:
                    plt.text(1, prop + 0.005, f"{round(prop * 100, 2)}%", color=color, fontweight="bold")

                x, y = ax.lines[i*2].get_data()
                ax.fill_between(x, y, where=y <= prop, color=color, alpha=0.1, hatch='xx')
                ax.fill_between(x, 0, prop, where=y >= prop, color=color, alpha=0.1, hatch='xx')
            plt.axvline(threshold, color="red", linestyle='dashed', lw="1")
            plt.text(threshold, -0.03, f"{threshold}km", color="red", fontweight="bold")

            box = ax.get_position()
            ax.set_position([box.x0, box.y0 + box.height * 0.1, box.width, box.height * 0.9])
            ax.legend(loc='upper center', title="Mean per outcome", bbox_to_anchor=(0.5, -0.1),
fancybox=True, shadow=True, ncol=2)

        plt.title(f'{self.prefix}\nCumulative distribution for distance error with {threshold} km
threshold')
        plt.xlabel('log Distance error (km)')
        plt.ylabel('Proportion')

        if save:
            filename = f"results/img/cum_dist_{self.prefix}_N{self.size}_{datetime.today().strftime('%Y-
%m-%d')}.png"
            plt.savefig(filename, dpi=300)
            print(
                f"VAL\tSAVE\tError distance cumulative distribution of {self.size} samples is drawn to
file: {filename}")
        if not self.cluster:
            plt.show()

    # distance error lines on the map
    def interactive_map(self, lines=True, best=False, size=1000, scope="world", save=True):
        size = min(self.size, size)
        self.df = self.df.sample(n=size, random_state=42, ignore_index=True)
        self.size = len(self.df.index)

        fig = go.Figure()

        # true points
```

```
        fig.add_trace(go.Scattergeo(
            lon=self.df["lon"], lat=self.df["lat"], mode='markers',
            marker=dict(
                size=2,
                color='rgb(0, 0, 0)',
            )))

        lon, lat, dist = [], [], []
        for o in range(self.outcomes):
            if best and o > 1:
                continue
            for i in range(self.size):
                lon.append(np.array(self.df.loc[i, f"O{o+1}_point"])[0])
                lat.append(np.array(self.df.loc[i, f"O{o+1}_point"])[1])
                dist.append(self.df.loc[i, f"O{o+1}_dist"])

        if lines:
            # outcome points
            fig.add_trace(go.Scattergeo(
                lon=lon, lat=lat, mode='markers',
                marker=dict(
                    size=2,
                    color="black"
                )
            ))
            # outcome lines
            for o in range(self.outcomes):
                if best and o > 1:
                    continue
                color = self.palette[o+1] if o < 5 else self.palette[5]
                for i in range(self.size):
                    fig.add_trace(go.Scattergeo(
                            lon=[self.df["lon"][i], np.array(self.df.loc[i, f"O{o+1}_point"])[0]],
                            lat=[self.df["lat"][i], np.array(self.df.loc[i, f"O{o+1}_point"])[1]],
                            mode='lines',
                            line=dict(
                                width=1,
                                color=color,
                            ),
                            hovertext=f'Distance: {round(self.df[f"O{o+1}_dist"][i], 2)} km'
                                      f'<br>{self.df[self.feature][i]}',
                            hoverinfo="name+text+lon+lat",
                            name="", opacity=0.5
                        )
                    )

            fig.update_layout(
                title_text=f'{self.prefix}<br>'
                           f'Error in distance between original and predicted geo locations of<br>'
                           f'{self.size} samples on {scope} scope, from model validated on {self.feature}
feature',
                showlegend=False,
                geo=dict(
                    scope=scope,
                    projection_type='natural earth',
                    showland=True,
                    landcolor='rgb(243, 243, 243)',
                    countrycolor='rgb(204, 204, 204)',
                ),
            )
        else:
            min_dist = np.log10(1e-10)
            max_dist = np.log10(20100)

            # outcome points
            fig.add_trace(go.Scattergeo(
                lon=lon, lat=lat, mode='markers',
                marker=dict(
                    size=5,
                    color=np.log10(dist),
                    colorscale="RdYlGn",
                    reversescale=True,
                    cmin=max_dist,
                    cmax=min_dist,
                    colorbar=dict(
                        orientation="h",
                        title="Log distance",
```

```
                        y=-0.1
                    )
                ),
                name="",
                hovertext=dist,
                hoverinfo="name+text+lon+lat",
            ))

            fig.update_layout(
                coloraxis_colorbar=dict(
                    title="Distance"),
                title_text=f'{self.prefix}<br>'
                           f'Error in distance between original and predicted geo locations of<br>'
                           f'{self.size} samples on {scope} scope, from model validated on {self.feature}
feature',
                showlegend=False,
                geo=dict(
                    scope=scope,
                    projection_type='natural earth',
                    showland=True,
                    landcolor='rgb(243, 243, 243)',
                    countrycolor='rgb(204, 204, 204)',
                ),
            )


        fig.update_geos(
            lataxis_showgrid=True, lonaxis_showgrid=True,
            visible=False, resolution=110,
            showcountries=True, countrycolor="Gray"
        )

        if save:
            filename = f"results/map-
html/intermap_{self.prefix}_N{self.size}_{datetime.today().strftime('%Y-%m-%d')}.html"
            fig.write_html(filename)
            print(f"VAL\tSAVE\tMap plot of {self.size} samples is drawn to file: {filename}")
        if not self.cluster:
            fig.show()
```

Файл regressor.py

```
import torch.nn as nn
from transformers import BertModel, BertPreTrainedModel, BertConfig

# general model wrapper
# linear regression fork for features and preset outputs
class BERTregModel():
    def __init__(self, n_outcomes=1, covariance=None, weighted=False, features=None, base_model_name=None,
hub_model=None):
        self.n_outcomes = n_outcomes
        self.cov = covariance
        self.weighted = weighted
        self.features = ["NON-GEO"] if features is None else features

        print(f"MODEL\tInitializing BERT Regression model for {self.n_outcomes} outcome(s)")
        # features
        print(f"MODEL\tText features:\t{' + '.join(self.features)}")
        # longitude, latitude for n outcomes
        self.coord_output = self.n_outcomes * 2
        print(f"MODEL\tCoordinates:\t{self.coord_output}")
        # weights of gaussians
        self.weights_output = self.n_outcomes if self.weighted and self.n_outcomes > 1 else 0
        if self.weights_output > 0:
            print(f"MODEL\tWeights:\t{self.weights_output}")

        # covariance matrix
        self.covariances = {'spher': self.n_outcomes,
                            'diag': self.n_outcomes * 2,
                            'tied': 3,
                            'full': self.n_outcomes * 3}
        if self.cov is None:
            self.cov_output = 0
            print(f"MODEL\tNon-probabilistic model has been chosen")
        else:
```

```
            if self.cov not in self.covariances:
                self.cov = 'spher'
            self.cov_output = self.covariances[self.cov]
            print(f"MODEL\tCovariances:\t{self.cov_output}\tmatrix type:\t{self.cov}")

        self.original_model = "bert-base-multilingual-cased" if base_model_name is None else
base_model_name
        print(f"MODEL\tOriginal model to load:\t{self.original_model}")

        self.key_output = self.coord_output + self.weights_output + self.cov_output
        self.minor_output = 2
        self.minor_output += 1 if self.cov_output > 0 else 0

        self.feature_outputs = {}
        for f in range(len(self.features)):
            if f == 0:
                output = self.key_output
                print(f"MODEL\tKey feature \t{self.features[f]} outputs:\t{output}")
            else:
                output = self.minor_output
                print(f"MODEL\tMinor feature\t{self.features[f]} outputs:\t{output}")
            self.feature_outputs[self.features[f]] = output

        if hub_model:
            self.model = GeoBertModel(BertConfig.from_pretrained(self.original_model),
self.feature_outputs)
            print(f"LOAD\tLoading HF model from {hub_model}")
            self.model = self.model.from_pretrained(hub_model, self.feature_outputs)
        else:
            self.model = BertRegressor(self.original_model, self.feature_outputs)



# Train model wrapper layer
class BertRegressor(nn.Module):
    def __init__(self, model_name, feature_outputs):
        super(BertRegressor, self).__init__()
        self.bert = BertModel.from_pretrained(model_name, return_dict=True)
        self.feature_outputs = feature_outputs

        self.key_regressor = nn.Linear(768, list(self.feature_outputs.values())[0])
        if len(self.feature_outputs) > 1:
            self.minor_regressor = nn.Linear(768, list(self.feature_outputs.values())[1])

    def forward(self, input_ids, attention_masks, feature_name):
        outputs = self.bert(input_ids, attention_masks)
        if feature_name == list(self.feature_outputs.keys())[0]:
            outputs = self.key_regressor(outputs[1])
        else:
            outputs = self.minor_regressor(outputs[1])
        return outputs


# HF model wrapper layer
class GeoBertModel(BertPreTrainedModel):
    def __init__(self, config, feature_outputs):
        super().__init__(config)
        self.bert = BertModel(config)
        self.feature_outputs = feature_outputs

        self.key_regressor = nn.Linear(config.hidden_size, list(self.feature_outputs.values())[0])
        if len(self.feature_outputs) > 1:
            self.minor_regressor = nn.Linear(config.hidden_size, list(self.feature_outputs.values())[1])

    def forward(self, input_ids, attention_mask=None, token_type_ids=None, position_ids=None,
head_mask=None, feature_name=None):
        outputs = self.bert(input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids,
position_ids=position_ids, head_mask=head_mask)
        pooler_output = outputs[1]
        if feature_name is None or feature_name == list(self.feature_outputs.keys())[0]:
            custom_output = self.key_regressor(pooler_output)
        else:
            custom_output = self.minor_regressor(pooler_output)
        return custom_output
```

Файл valid_data.py

```
from utils.result_visuals import *
from utils.regressor import *
import torch

# results manager and visual test on evaluated datasets
ww = "bert-base-multilingual-cased"
us = "bert-base-cased"

feature = "NON-GEO"
file = f"U-NON-GEO+GEO-ONLY-O1-d-total_mean-mf_mean-pos_spher-N30e5-B10-E3-cosine-LR[1e-05;1e-
06]_predicted_N300000_VF-NON-GEO_2023-02-19"

input_pred = f"results/val-data/{file}.jsonl"

# output_pred = f"results/val-data/{file}-out.jsonl"
# output_map_point = f"img/map-test-{feature}.png"
# output_map_line = f"img/map-test-{feature}.png"
# output_dist = f"img/dist-test-{feature}.png"

if torch.cuda.is_available():
    device = torch.device("cuda")
    print(f"Available GPU has {torch.cuda.device_count()} devices, using {torch.cuda.get_device_name(0)}")
else:
    print(f"No GPU available, using the CPU with {torch.get_num_threads()} threads instead.")
    device = torch.device("cpu")

bert_wrapper = BERTregModel(n_outcomes=1, covariance="spher", weighted=False, features=["NON-GEO", "GEO-
ONLY"], model_name=ww)
model = ModelBenchmark(bert_wrapper, distance=True, loss_prob="pos", mf_loss="mean", total_loss="mean")
result = ResultManager(None, None, feature, device, model, scaled=False, by_user=False, prefix=file)
result.load_df(input_pred)

# metrics
# result.result_metrics(True, 100)
# result.result_metrics(False, 100)

result.performance()

# visual = ResultVisuals(result)

# standard
# visual.density()
# visual.cum_dist(False, 161)

# GMM
# visual.summarize_prediction(1)
# visual.gaus_map()
# visual.prob_map_animation(228)

# visual.interactive_map(lines=False, best=True)

# result.save_df()
```

Файл input_entry.py

```
import argparse
from utils.prediction import *
from utils.regressor import *

# Entry point for prediction from text (model .pth files needed)

local_ww_models = {
    "gsop": "G-NON-GEO+GEO-ONLY-O1",
    "gmop": "G-NON-GEO+GEO-ONLY-O5",
    "psop": "P-NON-GEO+GEO-ONLY-O1",
    "pmop": "P-NON-GEO+GEO-ONLY-O5"
}

outcomes = 5  # 1 or 5
prob = True  # True or False

features = ["NON-GEO", "GEO-ONLY"]
```

```
text_example = "CIA and FBI can track anyone, and you willingly give the data away"

local = False
hub_model_prefix = "k4tel/geo-bert-multilingual"


def main():
    parser = argparse.ArgumentParser(description='Prediction of geolocations')
    parser.add_argument('-o', '--outcomes', type=int, default=outcomes, help="Number of outcomes (lomg,
lat) per tweet (default: 5)")
    parser.add_argument('-s', '--spat', action="store_true", help="Use geospatial model (default:
probabilistic)")
    parser.add_argument('-l', '--local', action="store_true", help="Use model stored locally")
    parser.add_argument('-m', '--model', type=str, default=None, help='Filename prefix of local model OR
HuggingFace repository link')
    parser.add_argument('-t', '--text', type=str, default=None, help='Text to process (max: 300 words)')
    args = parser.parse_args()

    weighted = args.outcomes > 1
    covariance = None if args.spat else "spher"

    if args.model:  # models/final/<prefix>.pth file; NOTE correct setup is needed
        prefix = args.model
    elif args.model is None and args.local:  # picking local model according to the setup
        if outcomes > 1:
            local_model_prefix = local_ww_models["gmop"] if args.spat else local_ww_models["pmop"]
        else:
            local_model_prefix = local_ww_models["gsop"] if args.spat else local_ww_models["psop"]

        prefix = local_model_prefix
    else:  # setup for P-NON-GEO+GEO-ONLY-O5
        weighted = True
        covariance = "spher"
        args.outcomes = 5
        args.spat = False
        prefix = hub_model_prefix

    # if not local - loading automatically on BERTregModel init
    model_wrapper = BERTregModel(args.outcomes, covariance, weighted, features, None, prefix) \
        if not args.local else BERTregModel(args.outcomes, covariance, weighted, features)

    # if local - loading automatically on ModelOutput init
    prediction = ModelOutput(model_wrapper, prefix, args.local)

    print(f"MODEL\tBERT geo regression model is ready, you can now predict location from the text (300
words max) "
          f"in a form of {'Gaussian distributions (lon, lat, cov)' if prob else 'coordinates (lon, lat)'}"
          f" with {outcomes} possible prediction outcomes.\nNOTE\tOutcomes that have very low weight won't
be displayed")

    text = args.text if args.text else input("Insert text: ")
    while text != "exit":
        if len(text) == 0:
            text = text_example
        if len(text.split()) < 300:
            result = prediction.prediction_output(text, filtering=True, visual=False)

            if args.outcomes > 1:
                ind = np.argwhere(np.round(result.weights[0, :] * 100, 2) > 0)
                significant = result.means[0, ind].reshape(-1, 2)
                weights = result.weights[0, ind].flatten()
            else:
                significant = result.means.reshape(-1, 2)
                weights = np.ones(1)

            sig_weights = np.round(weights * 100, 2)
            sig_weights = sig_weights[sig_weights > 0]

            print(f"RESULT\t{len(sig_weights)} significant prediction outcome(s):")

            for i in range(len(sig_weights)):
                point = f"lon: {'  lat: '.join(map(str, significant[i]))}"
                print(f"\tOut {i + 1}\t{sig_weights[i]}%\t-\t{point}")

        else:
            print(f"Number of words is above 300, unable to process.")
```

```
        text = args.text if args.text else input("Insert text: ")


if __name__ == "__main__":
    main()
```

Файл prediction.py

```
from transformers import BertTokenizer
import torch
import numpy as np
from pathlib import Path
from utils.twitter_dataset import *
from utils.result_visuals import *

# single text prediction wrapper
# preprocessing and result visual output
class ModelOutput():
    def __init__(self, wrapper, model_prefix, local=False):
        self.prefix = model_prefix
        self.device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
        self.model = wrapper.model.to(self.device)
        self.local = local

        if self.local:
            local_model = f"models/final/{self.prefix}.pth"
            print(f"LOAD\tLoading local model from {local_model}")
            if not Path(local_model).is_file():
                print(f"LOAD [ERROR] Unable to load local model: file {local_model} does not exist")

            state = torch.load(local_model) if torch.cuda.is_available() else torch.load(local_model,
map_location='cpu')
            self.model.load_state_dict(state['model_state_dict'])

        self.outcomes = wrapper.n_outcomes
        self.cov = wrapper.cov
        self.weighted = wrapper.weighted
        self.feature = wrapper.features[0]
        self.tokenizer = BertTokenizer.from_pretrained(wrapper.original_model)
        self.benchmark = ModelBenchmark(wrapper, True, "pos", "mean", "mean" if self.cov else "type")

        self.result = None
        self.visual = None

    def prediction_output(self, text, filtering=True, visual=False):
        if filtering:
            text = nlp_filtering(text)
            print(f"TEXT\tFiltered text: {text}")

        self.result = ResultManager(None, text, self.feature, self.device, self.benchmark, False, False,
self.prefix)

        if self.local:
            print("TEXT\tTokenizing text to input IDs and attention masks")
            encoded_corpus = self.tokenizer(text=text,
                                            add_special_tokens=True,
                                            padding='max_length',
                                            truncation='longest_first',
                                            max_length=300,
                                            return_attention_mask=True)
            input_id = encoded_corpus['input_ids']
            attention_mask = encoded_corpus['attention_mask']

            input = torch.tensor(input_id).to(self.device).reshape(1, -1)
            mask = torch.tensor(attention_mask).to(self.device).reshape(1, -1)

            self.model.eval()
            with torch.no_grad():
                output = self.model(input, mask, self.feature)

                if self.cov:
                    prob_model = self.benchmark.prob_models(output)

                output = output.cpu().numpy() if torch.cuda.is_available() else output.numpy()
```

```
            print(f"RESULT\tPost-processing raw model outputs: {output}")
            self.result.soft_outputs(list([prob_model])) if self.cov else
self.result.coord_outputs(output)

        else:
            print("TEXT\tTokenizing text to input IDs and attention masks")
            inputs = self.tokenizer(text, return_tensors="pt")

            with torch.no_grad():
                output = self.model(**inputs)
                prob_model = self.benchmark.prob_models(output)

            print(f"RESULT\tPost-processing raw model outputs: {output}")
            self.result.soft_outputs(list([prob_model]))

        if visual:
            self.visual = ResultVisuals(self.result)
            self.visual.text_map_result()

        return self.result
```

Файл buildDocker

```
#!/bin/bash
docker build --tag bert_bot .
```

Файл Dockerfile

```
FROM python:3.9
COPY . /app
WORKDIR /app/
# RUN apt update
# RUN apt install -y python3.9 python3.9-dev python3.9-venv python3-pip python3-wheel build-essential
RUN python3.9 -m pip install --upgrade pip
RUN pip3 install -r requirements.txt
```

Файл runDocker

```
#!/usr/bin/env sh
docker run \
--privileged \
-v $(pwd) -it --rm --entrypoint="" \
bert_bot python3.9 bot_src.py
```

Файл runBot.py

```
#!/usr/bin/env sh
tmux new -s bert_bot "bash runDocker"
```

Файл bot_src.py

```
#!/bin/python
import requests
import sqlite3
import os.path
from time import sleep
import json
```

```python
from bert_predict import *

BOT_TOKEN = ""
BASE_URL = f"https://api.telegram.org/bot{BOT_TOKEN}/"
PATH_TO_DB = "db.db"

START = "You can use this bot to access geolocation prediction model:\n" \
        "/predict _<text>_ (3-500 words) - get prediction results\n" \
        "/info - get information about the model"

INFO = "This model predicts the geolocation of short texts (less than 500 words) in a form of " \
       "two-dimensional distributions also referenced as the Gaussian Mixture Model (GMM).\n" \
       "\nBERT Regression model for 5 outcome(s):\nCoordinates: 10\nWeights: 5\nCovariances: 5 (matrix
type: spher)\n" \
       "\n\U0001F464 [GitHub project repo](https://github.com/K4TEL/geo-twitter.git)\n" \
       "\n\U0001F917 [HuggingFace model repo](https://huggingface.co/k4tel/geo-bert-multilingual)\n" \
       "\n\U0001F4F0 [arXiv paper preprint](https://arxiv.org/pdf/2303.07865.pdf)"


# parsing functions
# user message parsing
def parse_cmd(msg):
    text = msg["message"]["text"]
    if len(text) == 0:  # empty text
        return None, None
    if text[0] != "/":  # not a command
        return None, text
    text = text[1:]  # crop "/"
    if " " in text:  # command with an argument
        split = text.split(" ")
        return split[0], " ".join(split[1:])
    else:  # command without an argument
        return text, None


# user profile parsing
def parse_user(msg):
    user = msg["message"]["from"]
    return user["username"], f'{user["first_name"]} {user["last_name"]}'


# bot updates parsing
def parse_updates(update_id=0):  # -> new update_id, List[postDict]
    result = requests.post(f'{BASE_URL}getUpdates?update_id={update_id}').json()
    if result["ok"]:
        msgs = [item for item in result["result"] if item["update_id"] > update_id]  # get unprocessed
        if len(msgs) != 0:
            update_id = max(map(lambda msg: msg.get('update_id', 0), msgs))  # update last update_id
        return msgs, update_id
    return None


# DB functions:
# load DB connection
def load_DB(db_path):  # -> DB Connection
    is_created = os.path.exists(db_path)
    try:
        conn = sqlite3.connect(db_path)
    except Exception as e:
        print(e)
        return None
    # create db if needed
    # columns: update_id, username, user tag, command, argument/text, bot response)
    if not is_created:
        conn.execute('''CREATE TABLE LOGS
                 (ID      INT  PRIMARY KEY NOT NULL,
                  USER    TEXT             NOT NULL,
                  TAG     TEXT             NOT NULL,
                  CMD     TEXT             NULL,
                  TEXT    TEXT             NULL,
                  OUT     TEXT             NULL);''')
    return conn


# log request and response to DB
def log_to_DB(conn, msg, response):
```

```
    usertag, username = parse_user(msg)
    command, arg = parse_cmd(msg)
    update_id = msg["update_id"]

    if conn is not None:
        cur = conn.cursor()
        try:
            sql_umid = ''' INSERT INTO LOGS(ID, USER, TAG, CMD, TEXT, OUT)
              VALUES(?,?,?,?,?,?) '''
            cur.execute(sql_umid, (update_id, usertag, username, command, arg, response))
            conn.commit()
            print(f"Message {update_id} from {usertag} was logged in DB")
        except Exception as e:
            print("Some kind of error occurred")
            print(e)


class BotChat():
    def __init__(self, chat_id, db_conn, model):
        self.chat_id = chat_id
        self.db_conn = db_conn
        self.model = model

        self.text_base_url = f"{BASE_URL}sendMessage?chat_id={self.chat_id}&parse_mode=Markdown"
        self.image_base_url = f"{BASE_URL}sendPhoto?chat_id={self.chat_id}&parse_mode=Markdown"

        self.handlers = {
            "start": self.handle_start,
            "info": self.handle_info,
            "predict": self.handle_predict,
        }

    # send text to chat
    def send_text(self, text):
        result = requests.get(f"{self.text_base_url}&text={text}").json()
        return result["result"] if result["ok"] else None

    # send image file with text caption to chat
    def send_image(self, filename, caption=""):
        result = requests.post(f"{self.image_base_url}&caption={caption}",
                               files={"photo": open(filename, "rb")}).json()
        return result["result"] if result["ok"] else None

    # routing user request to the command handlers
    def process_request(self, msg):
        command, arg = parse_cmd(msg)
        print(self.chat_id, command, arg)
        if command in self.handlers:
            self.handlers[command](arg, msg)
        else:
            self.send_text("Unknown command!")

    # /info handler
    def handle_info(self, arg, msg):
        if self.send_text(INFO):
            log_to_DB(self.db_conn, msg, "info")

    # /start handler
    def handle_start(self, arg, msg):
        if self.send_text(START):
            log_to_DB(self.db_conn, msg, "start")

    # /predict <text> handler
    def handle_predict(self, arg, msg):
        if not arg or len(arg) == 0:
            response = "Text argument is empty!"
            if self.send_text(response):
                log_to_DB(self.db_conn, msg, response)

        elif len(self.model.tokenizer.tokenize(arg)) > 512:
            response = "Text length exceeds 512 tokens!"
            if self.send_text(response):
                log_to_DB(self.db_conn, msg, response)

        else:
            self.send_text("Estimating geolocation and generating GMM plot...")
            result = text_prediction(self.model, arg)
```

```
            response = result.result_to_text()
            filename = f"{msg['update_id']}.png"

            visual = ResultVisuals(result)
            visual.text_map_result(filename)

            if self.send_image(filename, response):
                log_to_DB(self.db_conn, msg, response)
                os.remove(filename)


# run the bot
def bot_loop():
    model = load_model()
    conn_DB = load_DB(PATH_TO_DB)
    chats = {}

    if conn_DB is not None:
        cur = conn_DB.cursor()
        a = cur.execute("""SELECT MAX(ID) FROM LOGS;""")
        row = cur.fetchone()

    update_id = int(row[0]) if row and row[0] else 779342409

    print("Listening...")
    while conn_DB is not None:
        try:
            msgs, update_id = parse_updates(update_id)
            if msgs:
                print(msgs)
                for msg in msgs:
                    chat_id = msg["message"]["chat"]["id"]
                    if chat_id not in chats:
                        chats[chat_id] = BotChat(chat_id, conn_DB, model)

                    chat = chats[chat_id]
                    chat.process_request(msg)
        except Exception as e:
            print(e)
        finally:
            sleep(1)


if __name__ == "__main__":
    bot_loop()
```

Файл bert_predict.py

```
#!/bin/python
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

import torch
import torch.distributions as dist

import numpy as np

import torch.nn as nn
from transformers import BertModel, BertPreTrainedModel, BertConfig, BertTokenizer

import string
import re

from transformers import Pipeline, pipeline
from transformers.pipelines import PIPELINE_REGISTRY

hub_model = 'k4tel/geo-bert-multilingual'
base_model = "bert-base-multilingual-cased"


# general model wrapper
# linear regression fork for features and preset outputs
```

```
class BERTregModel():
    def __init__(self):
        self.n_outcomes = 5
        self.cov = "spher"

        print(f"MODEL\tInitializing BERT Regression model for {self.n_outcomes} outcome(s)")
        # features
        self.features = ["NON-GEO", "GEO-ONLY"]
        print(f"MODEL\tText features:\t{' + '.join(self.features)}")
        # longitude, latitude for n outcomes
        self.coord_output = self.n_outcomes * 2
        print(f"MODEL\tCoordinates:\t{self.coord_output}")
        # weights of gaussians
        self.weights_output = self.n_outcomes
        print(f"MODEL\tWeights:\t{self.weights_output}")
        # covariance of gaussians
        self.cov_output = self.n_outcomes
        print(f"MODEL\tCovariances:\t{self.cov_output}\tmatrix type:\t{self.cov}")
        # base model
        self.original_model = "bert-base-multilingual-cased"
        print(f"MODEL\tOriginal model to load:\t{self.original_model}")

        self.key_output = self.coord_output + self.weights_output + self.cov_output
        self.minor_output = 2
        self.minor_output += 1 if self.cov_output > 0 else 0

        self.feature_outputs = {}
        for f in range(len(self.features)):
            if f == 0:
                output = self.key_output
                print(f"MODEL\tKey feature \t{self.features[f]} outputs:\t{output}")
            else:
                output = self.minor_output
                print(f"MODEL\tMinor feature\t{self.features[f]} outputs:\t{output}")
            self.feature_outputs[self.features[f]] = output

        self.outputs_map = {
            "coord": [0, self.coord_output],
            "weight": [self.coord_output,
                       self.coord_output + self.weights_output],
            "sigma": [self.coord_output + self.weights_output,
                      self.coord_output + self.weights_output + self.cov_output] if self.cov else None
        }

        self.model = GeoBertModel(BertConfig.from_pretrained(self.original_model), self.feature_outputs)
        self.tokenizer = BertTokenizer.from_pretrained(self.original_model)


# HF model wrapper layer
class GeoBertModel(BertPreTrainedModel):
    def __init__(self, config, feature_outputs):
        super().__init__(config)
        self.bert = BertModel(config)
        self.feature_outputs = feature_outputs

        self.key_regressor = nn.Linear(config.hidden_size, list(self.feature_outputs.values())[0])
        self.minor_regressor = nn.Linear(config.hidden_size, list(self.feature_outputs.values())[1])

    def forward(self, input_ids, attention_mask=None, token_type_ids=None, position_ids=None,
head_mask=None, feature_name=None):
        outputs = self.bert(input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids,
position_ids=position_ids, head_mask=head_mask)
        pooler_output = outputs[1]
        if feature_name is None or feature_name == list(self.feature_outputs.keys())[0]:
            custom_output = self.key_regressor(pooler_output)
        else:
            custom_output = self.minor_regressor(pooler_output)
        return custom_output


# torch.nn.Softplus() with a lower bound in munpy
def softplus(x, threshold=20):
    x_clipped = np.clip(x, -threshold, threshold)
    result = np.log(1 + np.exp(x_clipped))
    result[x > threshold] = x[x > threshold]
    return result + (1 / (2 * np.pi))
```

```
# torch.nn.Softmax() in munpy
def softmax(x, axis=-1):
    x_max = np.max(x, axis=axis, keepdims=True)
    x_exp = np.exp(x - x_max)
    x_sum = np.sum(x_exp, axis=axis, keepdims=True)
    return x_exp / x_sum


# filtering
def filter_text(text):
    print(f"TEXT\tFiltering text: {text}")
    pattern = r'http\S+'
    text = re.sub(pattern, '', text)
    text = "".join([i for i in text if i not in string.punctuation])
    return text


# custom HF pipeline
class GeoRegressorPipeline(Pipeline):
    def _sanitize_parameters(self, **kwargs):
        preprocess_kwargs = {}
        if "filter" in kwargs:
            preprocess_kwargs["filter"] = kwargs["filter"]

        postprocess_kwargs = {}
        if "outputs_map" in kwargs:
            postprocess_kwargs["outputs_map"] = kwargs["outputs_map"]

        return preprocess_kwargs, {}, postprocess_kwargs

    def preprocess(self, text, filter=True):
        text = filter_text(text) if filter else text
        return self.tokenizer(text, return_tensors=self.framework)

    def _forward(self, model_inputs):
        return self.model(**model_inputs)

    def postprocess(self, model_outputs, outputs_map=None):
        print(f"RESULT\tPost-processing raw model outputs: {model_outputs}")
        model_outputs = model_outputs.numpy() if self.device == "cpu" else model_outputs.cpu().numpy()

        outcomes = outputs_map["weight"][1] - outputs_map["weight"][0]

        P = model_outputs[0, outputs_map["coord"][0]:outputs_map["coord"][1]]
        means = P.reshape([outcomes, 2])

        W = model_outputs[0, outputs_map["weight"][0]:outputs_map["weight"][1]]
        weights = softmax(W, axis=0).reshape(outcomes)

        if outputs_map["sigma"]:
            S = model_outputs[0, outputs_map["sigma"][0]:outputs_map["sigma"][1]]
            S = softplus(S)
            sigma = np.eye(2).reshape(1, 2, 2) * S.reshape(-1, 1, 1)
            covs = sigma.reshape([outcomes, 2, 2])

        print(f"RESULT\tSorting all outputs for {outcomes} outcomes by probabilistic weights")
        sort_indexes = np.argsort(weights)
        index = sort_indexes[::-1]

        means, weights, covs = means[index], weights[index], covs[index]

        result = []
        for i in range(5):
            result.append({"point": means[i], "weight": weights[i], "cov": covs[i]})

        return result


# result manager
class ResultManager():
    def __init__(self, text, feature, device, model, prefix=None):
        self.cluster = device.type == "cuda"
        self.feature = feature
        if prefix is None:
            prefix = feature
        self.prefix = prefix
```

```
        self.model = model

        self.outcomes = self.model.n_outcomes

        self.text = text

        self.means = None
        self.weights = None
        self.covs = None

    # pipeline result to params
    def pipeline_to_params(self, result_dict):
        self.means = np.zeros((1, self.outcomes, 2), float)
        self.weights = np.zeros((1, self.outcomes), float)
        self.covs = np.zeros((1, self.outcomes, 2, 2), float)

        for i in range(self.outcomes):
            self.means[0, i, :] = result_dict[i]["point"]
            self.weights[0, i] = result_dict[i]["weight"]
            self.covs[0, i, :] = result_dict[i]["cov"]

    # result to string
    def result_to_text(self):
        ind = np.argwhere(np.round(self.weights[0, :] * 100, 2) > 0)
        significant = self.means[0, ind].reshape(-1, 2)
        weights = self.weights[0, ind].flatten()
        sig_weights = weights[weights > 0]

        result = f"*TEXT:* {self.text}\n*RESULT:*"

        print(f"RESULT\t{len(sig_weights)} significant prediction outcome(s):")
        for i in range(len(sig_weights)):
            point = f"lon: {'  lat: '.join(map(str, significant[i]))}"
            weight = str(np.round(sig_weights[i] * 100, 2))
            result += f"\nPrediction outcome {i + 1}  -  {weight}%\n" \
                      f"*lon:* `{str(significant[i][0])}`\n*lat:* `{str(significant[i][1])}`"
            print(f"\tOut {i + 1}\t{weight}%\t-\t{point}")

        return result


# GMM
def dist_gmm(outcomes, means, covs, weights):
    means, covs = means.reshape(outcomes, 2), covs.reshape(outcomes, 2, 2)
    gaussian = dist.MultivariateNormal(torch.from_numpy(means), torch.from_numpy(covs))
    gmm_weights = dist.Categorical(torch.from_numpy(weights.reshape(-1)))
    gmm = dist.MixtureSameFamily(gmm_weights, gaussian)
    return gmm


# generating map grid with intergrid peaks
def map_grid(peaks, step=10):
    xmin, xmax = -180, 180
    ymin, ymax = -90, 90
    x = np.linspace(xmin, xmax, step)
    y = np.linspace(ymin, ymax, step)
    x = np.concatenate((x, peaks[:, 0]), axis=0)
    x = np.sort(x)
    y = np.concatenate((y, peaks[:, 1]), axis=0)
    y = np.sort(y)
    xx, yy = np.meshgrid(x, y)
    return xx, yy


# visualization of results
class ResultVisuals():
    def __init__(self, manager):
        self.manager = manager
        self.cluster = self.manager.cluster
        self.feature = self.manager.feature
        self.prefix = self.manager.prefix

        self.outcomes = self.manager.outcomes

        # palette for sorted by weight outcomes
```

```
        self.palette = {1: 'darkgreen',
                        2: 'goldenrod',
                        3: 'darkorange',
                        4: 'crimson',
                        5: 'darkred'}

    # GMM plots
    def plot_gmm(self, means, covs, weights, title, filename):
        fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 15))

        xmin, xmax = -180, 180
        ymin, ymax = -90, 90
        step_big = 45.
        ticks_big_x, ticks_big_y = range(int(xmin), int(xmax), int(step_big)), \
                                   range(int(ymin), int(ymax), int(step_big))
        tick_labels_big_x, tick_labels_big_y = [str(x) for x in ticks_big_x], \
                                               [str(y) for y in ticks_big_y]

        total_peaks = means.reshape(-1, 2)
        peaks_unique, ind = np.unique(np.round(total_peaks, 4), return_index=True, axis=0)
        intergrid_peaks = total_peaks[ind]
        gmm = dist_gmm(self.outcomes, means, covs, weights)

        # BIG MAP
        grid_step = 400
        xxb, yyb = map_grid(intergrid_peaks, grid_step)
        XX_big = np.array([xxb.ravel(), yyb.ravel()]).T

        Z_big = gmm.log_prob(torch.from_numpy(XX_big)).numpy()

        ind = np.argwhere(np.round(weights * 100, 2) > 0)
        significant = means[ind].reshape(-1, 2)
        sig_weights = weights[ind].flatten()

        margin_lon, margin_lat = 10, 5
        min_lon, max_lon = min(significant[:, 0]) - margin_lon, max(significant[:, 0]) + margin_lon
        min_lat, max_lat = min(significant[:, 1]) - margin_lat, max(significant[:, 1]) + margin_lat

        # ZOOM MAP
        step_zoom = 15.0
        ticks_zoom_x, ticks_zoom_y = range(int(min_lon), int(max_lon), int(step_zoom)), \
                                     range(int(min_lat), int(max_lat), int(step_zoom))
        tick_labels_zoom_x, tick_labels_zoom_y = [str(x) for x in ticks_zoom_x], \
                                                 [str(y) for y in ticks_zoom_y]
        xxz, yyz = np.mgrid[min_lon:max_lon:400j, min_lat:max_lat:400j]
        XX_zoom = np.array([xxz.ravel(), yyz.ravel()]).T

        Z_zoom = np.exp(gmm.log_prob(torch.from_numpy(XX_zoom)).numpy())

        # MAPS
        Z_big, Z_zoom = Z_big.reshape(xxb.shape), Z_zoom.reshape(xxz.shape)
        zbmin, zzmin = np.min(Z_big), np.min(Z_zoom)
        zbmax, zzmax = np.max(Z_big), np.max(Z_zoom)

        # BIF MAP
        ax_big = ax[0]

        ax_big.set_xlim(xmin, xmax)
        ax_big.set_ylim(ymin, ymax)
        ax_big.set_title(f'Log-Likelihood score of GMM', y=1.0, pad=24)

        map_big = Basemap(ax=ax_big, projection='mill', resolution='l')
        map_big.drawcoastlines(linewidth=0.5, color="black", zorder=2)
        map_big.drawcountries(linewidth=0.7, color="black", zorder=3)
        map_big.drawparallels(np.arange(ymin, ymax, step_big), labels=tick_labels_big_y)
        map_big.drawmeridians(np.arange(xmin, xmax, step_big), labels=tick_labels_big_x)
        map_big.drawmapboundary(fill_color='lightgrey', zorder=0)
        map_big.fillcontinents(color='white', lake_color='lightgrey', zorder=1)

        contour_big = map_big.contourf(xxb, yyb, Z_big, levels=np.linspace(zbmin, zbmax, 250),
                                       cmap='Spectral_r', alpha=0.7, zorder=9, latlon=True)
        plt.colorbar(contour_big, ax=ax_big, orientation="horizontal", pad=0.2)

        # ZOOM MAP
        ax_zoom = ax[1]
        ax_zoom.set_xlim(min_lon, max_lon)
        ax_zoom.set_ylim(min_lat, max_lat)
```

```
        ax_zoom.set_title('Max Probability Density Function region', y=1.0, pad=24)

        map_zoom = Basemap(ax=ax_zoom, projection='mill',
                           llcrnrlat = min_lat,
                           llcrnrlon = min_lon,
                           urcrnrlat = max_lat,
                           urcrnrlon = max_lon, resolution='h')
        map_zoom.drawcoastlines(linewidth=0.5, color="black", zorder=2)
        map_zoom.drawcountries(linewidth=0.7, color="black", zorder=3)
        map_zoom.drawmapboundary(fill_color='lightgrey', zorder=0)
        map_zoom.drawparallels(np.arange(min_lat, max_lat, step_zoom), labels=tick_labels_zoom_y)
        map_zoom.drawmeridians(np.arange(min_lon, max_lon, step_zoom), labels=tick_labels_zoom_x)
        map_zoom.fillcontinents(color='white', lake_color='lightgrey', zorder=1)

        Z_zoom = np.ma.array(Z_zoom, mask=Z_zoom < 1e-8)
        contour_zoom = map_zoom.contourf(xxz, yyz, Z_zoom, levels=np.linspace(zzmin, zzmax, 250),
                                         cmap='Spectral_r', alpha=0.7, zorder=9, latlon=True)
        plt.colorbar(contour_zoom, ax=ax_zoom, orientation="horizontal", pad=0.2)

        for i in range(len(sig_weights)):
            color = self.palette[i+1] if i < 5 else self.palette[5]
            point = f"lon: {'  lat: '.join(map(str, significant[i])) }"
            label = point + f" - {round(sig_weights[i] * 100, 2)}%",
            map_zoom.scatter(significant[i, 0], significant[i, 1], latlon=True,
                             label=label, color=color,
                             s=10, zorder=9999)
            map_zoom.scatter(significant[i, 0], significant[i, 1], latlon=True,
                             color=color, alpha=0.2,
                             s=100, zorder=9999)

        plt.legend(loc='upper center', title="Predicted outcomes", bbox_to_anchor=(0.5, -0.1),
fancybox=True, shadow=True)
        plt.suptitle(title)

        fig.savefig(filename)

    # single text results visualization on the map
    def text_map_result(self, filename):
        means = self.manager.means[0]
        weights = self.manager.weights[0]
        title = f'{self.prefix}\nplots of GMM with {self.outcomes} means'
        if self.manager.text:
            title += f"\nText: {self.manager.text}\n"

        covs = self.manager.covs[0]
        self.plot_gmm(means, covs, weights, title, filename)


# load model or model pipeline
def load_model():
    model_wrapper = BERTregModel()
    model_wrapper.tokenizer = BertTokenizer.from_pretrained(hub_model)
    model_wrapper.prefix = hub_model

    PIPELINE_REGISTRY.register_pipeline(
        "geo-regressor",
        pipeline_class=GeoRegressorPipeline,
        pt_model=GeoBertModel,
        default={"pt": (hub_model, "main")},
        type="text",
    )

    print(f"LOAD\tLoading HF model from {hub_model}")
    model_wrapper.pipeline = pipeline("geo-regressor",
                                      model=hub_model,
                                      tokenizer=model_wrapper.tokenizer,
                                      device="cuda" if torch.cuda.is_available() else "cpu",
                                      outputs_map=model_wrapper.outputs_map,
                                      filter=filter,
                                      model_kwargs={"feature_outputs": model_wrapper.feature_outputs})
    return model_wrapper


# get prediction result for a single text
def text_prediction(model_wrapper, text):
    result = ResultManager(text, "NON-GEO", torch.device("cuda") if torch.cuda.is_available() else
torch.device("cpu"),
```

```
                            model_wrapper, model_wrapper.prefix)
    outputs = model_wrapper.pipeline(text)
    result.pipeline_to_params(outputs)
    return result


def main():
    model_wrapper = load_model(base_model, hub_model)

    text = "CIA and FBI can track anyone, and you willingly give the data away"

    result = text_prediction(model_wrapper, text)

    ind = np.argwhere(np.round(result.weights[0, :] * 100, 2) > 0)
    significant = result.means[0, ind].reshape(-1, 2)
    weights = result.weights[0, ind].flatten()

    sig_weights = weights[weights > 0]

    print(f"RESULT\t{len(sig_weights)} significant prediction outcome(s):")
    for i in range(len(sig_weights)):
        point = f"lon: {'  lat: '.join(map(str, significant[i]))}"
        weight = str(np.round(sig_weights[i] * 100, 2))
        print(f"\tOut {i + 1}\t{weight}%\t-\t{point}")

    # visual = ResultVisuals(result)
    # visual.text_map_result("filename.png")


if __name__ == "__main__":
    main()
```

Файл requirements.txt

```
geopandas==0.12.2
geopy==2.3.0
GPUtil==1.4.0
imageio==2.25.1
matplotlib==3.7.0
moviepy==1.0.3
numpy==1.21.5
pandas==1.5.2
plotly==5.13.1
psutil==5.9.4
pyarrow==11.0.0
scikit_learn==1.2.1
scipy==1.10.1
seaborn==0.12.2
Shapely==2.0.1
spacy==3.5.0
torch==1.13.1
torchvision==0.14.1
torchaudio==0.13.1
torchtext==0.14.1
fastai==2.7.11
tokenizers
torchdata==0.5.1
tqdm==4.64.1
transformers==4.26.0
Basemap==1.3.6
basemap-data-hires==1.3.2
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Алгоритм та програмне забезпечення для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT

Програма та методика тестування

КПІ.ІТ-9116.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ігор БАКЛАН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Катерина ЛУЦАЙ

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є ПЗ Розробника та серверна частина інтерфейсу Користувача у вигляді Телеграм боту.

В рамкам оцінки якості ПЗ Розробка основним об’єктом дослідження виступає модель нейронної мережі для прогнозування місцезнаходження за текстом. Тестування та оцінка моделі відбувається за загальноприйнятими метриками використаними у попередніх роботах спрямованих на вирішення задачі прогнозування місцезнаходження у форматі пари координат або параметрів двовимірного розподілу на мапі. Для тестування функціоналу ПЗ Розробника (навчання, оцінка, прогноз моделі) було проведено мануальні тести.

Для тестування роботи Телеграм боту було проведено лише мануальне тестування функціоналу оскільки це інтерфейс слугує лише для демонстрації найкращої з навчених моделей.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження проміжних та фінальних даних;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування. Реалізовано завдяки сервісу embold;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми. Реалізовано мануально;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній. Реалізовано мануально;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних нейронної мережі. Автоматизовано підрахунок метрик за результатами оцінки моделей на нобрах даних;
- тестування «сірої скриньки» – об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій). Автоматизовано моніторинг метрик навчання (втрат, точності, тощо).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування на виведення повідомлень про помилку, коли це необхідно;
- тестування інтерфейсу користувача;
- тестування зручності використання;
- тестування алгоритмів машинного навчання на точність прогнозів місцезнаходження при заданих вхідних змінних;

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Алгоритм та програмне забезпечення для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT

Керівництво користувача

КПІ.ІТ-9116.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ігор БАКЛАН

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Катерина ЛУЦАЙ

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

GeoBERT – це проект спрямований на розв'язання задачі прогнозування геолокації твітів/користувачів та надання гнучкої методології для геотегування великих масивів текстових даних. Запропонований підхід реалізує нейронні мережі для обробки природної мови (NLP) для оцінки місцезнаходження у вигляді координат (довгота, широта) і двовимірних моделей гауссової суміші (GMM). Дослідження запропонованих модифікацій архітектури та робота алгоритмів машинного навчання була перевірена на наборі даних Twitter з використанням попередньо навчених двонаправлених представлень кодерів з трансформерів (BERT) як базових моделей.

З метою демонстрації найкращої з навчених моделей задачі цього проекту включали надання відкритого доступу користувачам Telegram до використання розробленої моделі прогнозування місцезнаходження. Данна модель була доопрацьована на всесвітньому наборі даних Twitter з використанням попередньо навченої багатомовної BERT в якості базової моделі – NG+GO_PMOP_5wLBSP – моделі багатозадачного навчання (NON-GEO як KF та GEO-ONLY як MF) з обмеженим параметром коваріації для вихідних параметрів GMM з 5 центрами (точками / результатами).

Для Розробника надано набір алгоритмів та демонстрацію в рамках консольного інтерфейсу що за структурою поділений на два модулі: навчання та оцінка локальних моделей на власних датасетах, прогнозування місцезнаходження за текстом з прикладом у Jupyter Notebook що використовує модель на HuggingFace – geo-bert-multilingual – репозиторій на HuggingFace найкращої моделі (ймовірнісна, 5 результатів, NON-GEO + GEO-ONLY), навченої на світовому наборі даних Twitter

Далі наведено структуру модуля навчання та оцінки:

datasets – тека з вихідними файлами вхідних наборів даних, що використовуються під час навчання та оцінювання. Для коректного читання файли повинні мати формат .jsonl, що містить стовпці "lon", "lat", "text", "user" та "place" (поля JSON).

models – тека з файлами локальних моделей та чекпоінтів у форматі .pth.

results – тека для вихідних файлів, таких як зображення, оцінені набори даних та звіти метрик продуктивності.

utils – тека з важливими утилітами python

– *benchmark.py* – обчислення функції втрат і робота з журналом метрик навчання Tensorboard

– *twitter_dataset.py* – клас-обгортка набору даних, що реалізує формування ознак, токенізацію та створення загружчиків даних PyTorch

– *regressor.py* – шар обгортки лінійної регресії для базових моделей BERT

– *result_manager.py* – постобробка результатів моделювання, запис та читання .jsonl-файлів результатів оцінювання, обчислення метрик продуктивності

– *result_visuals.py* – візуалізація результатів на графіках методами Matplotlib

– *prediction.py* – прогнозування одного тексту

– *model_trainer.py* – навчання та оцінка моделей

У кореневій теці проекту розміщено наступні скрипти:

train_bert.py – введення параметрів командного рядка, точка входу для навчання та оцінювання

input_entry.py – точка входу для прогнозування за окремим текстом з використанням моделей локального або HF-репозиторію

Додатково надано:

runs – папка для зберігання лог-файлів навчання Tensorboard

scripts appendix – папка, що містить тестові та розробницькі python-скрипти, а також bash-скрипти для запуску завдань на кластері з системою управління slurm

valid_data.py – точка входу для управління та візуалізації статистики датасету результатів

collector.py – парсинг файлів бази даних Twitter для формування файлів датасетів

При використанні готової моделі з репозиторію HuggingFace для прогнозування за текстом надано наступні компоненти:

text_result.py – завантаження моделі з HF repo та постобробка результатів однотекстового прогнозу

prediction.ipynb – Jupyter Notebook для завантаження проекту та виконання прогнозування, Google Colab є копією файлу prediction.ipynb і доступний для запуску.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність високопродуктивного кластеру з операційною системою на базі ядра Unix – серверах без графічного інтерфейсу користувача;

– наявність доступу до Інтернету;

– для встановлення додатку на пристрої повинно бути не менше 10 Гб вільної пам’яті;

– для проведення навчання на пристрої рекомендовано не менше 12 Гб вільної відеопам’яті.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;

– об‘єм ОЗП: 4 Гб;

– підключення до мережі Інтернет зі швидкістю від 20 мегабіт;

– об’єм пам’яті: 10 Гб.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7;

– об‘єм ОЗП: 32 Гб;

– підключення до мережі Інтернет зі швидкістю від 100 мегабіт;

– об’єм пам’яті: 1 Тб;

– об’єм відеопам’яті: 12 Гб;

– тип графічної карти: NVIDIA GeForce GTX 1080 Ti.

2.2 Завантаження застосунку

Якщо ви Розробник, щоб запустити ПЗ локально, ви можете клонувати цей проект за допомогою:

```
& git clone -b main https://github.com/K4TEL/geo-twitter.git
```

Для навчання або оцінки на власних датасетах, або для прогнозування місцезнаходження за однотекстовими вхідними даними:

```
& git clone -b predict https://github.com/K4TEL/geo-twitter.git
```

Якщо ви звичайний Користувач, демонстрація роботи найкращої з навчених моделей світового рівня доступно до прогнозування місцезнаходження на власному тексті користувача при умовах роботи серверної частини Телеграм боту – основного інтерфейсу. Для запуску застосунку спершу необхідно відкрити Телеграм та знайти бота @geobertbot, щоб використовувати модель прогнозування місцезнаходження за командою «/predict».

2.3 Перевірка коректної роботи

Для Розробника, переконайтеся що у вас встановлена версія Python 3.8 або вище, а версія pip остання з можливих. Потім, у вашому віртуальному середовищі Python виконайте команду:

```
& pip install -r requirements.txt
```

В результаті правильного завершення встановлення залежних бібліотек Python у вас повинно бути готове до роботи середовище.

Для Користувача, якщо команда «/start» для запуску боту не дає відповіді, то інтерфейс демонстрації тимчасово не доступний. Пропонується скористатись інтуїтивно зрозумілим у використанні блокнотом Google Colab.

3 ВИКОНАННЯ ПРОГРАМИ

Для звичайного Користувача – Telegram-бот моделі прогнозування місцезнаходження за текстом GeoBERT

Доступні три команди:

/start – запустити бота як показано на Рисунку 3.1

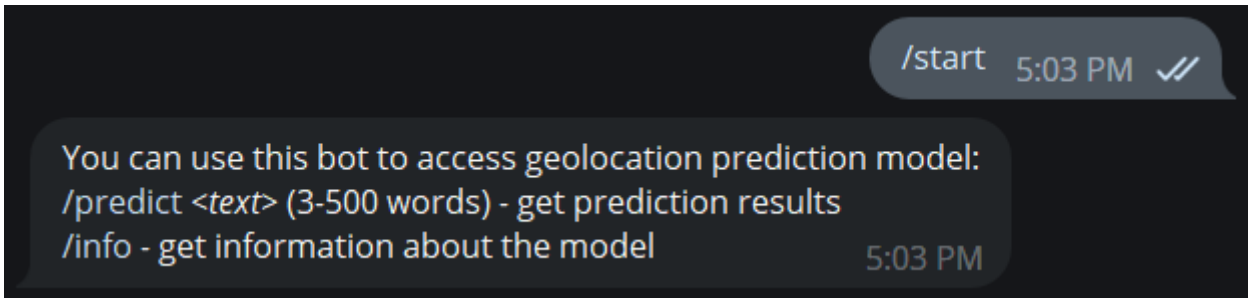


Рисунок 3.1 – Початкова відповідь бота на команду “/start”

/info – отримати інформацію про модель як показано на Рисунку 3.2

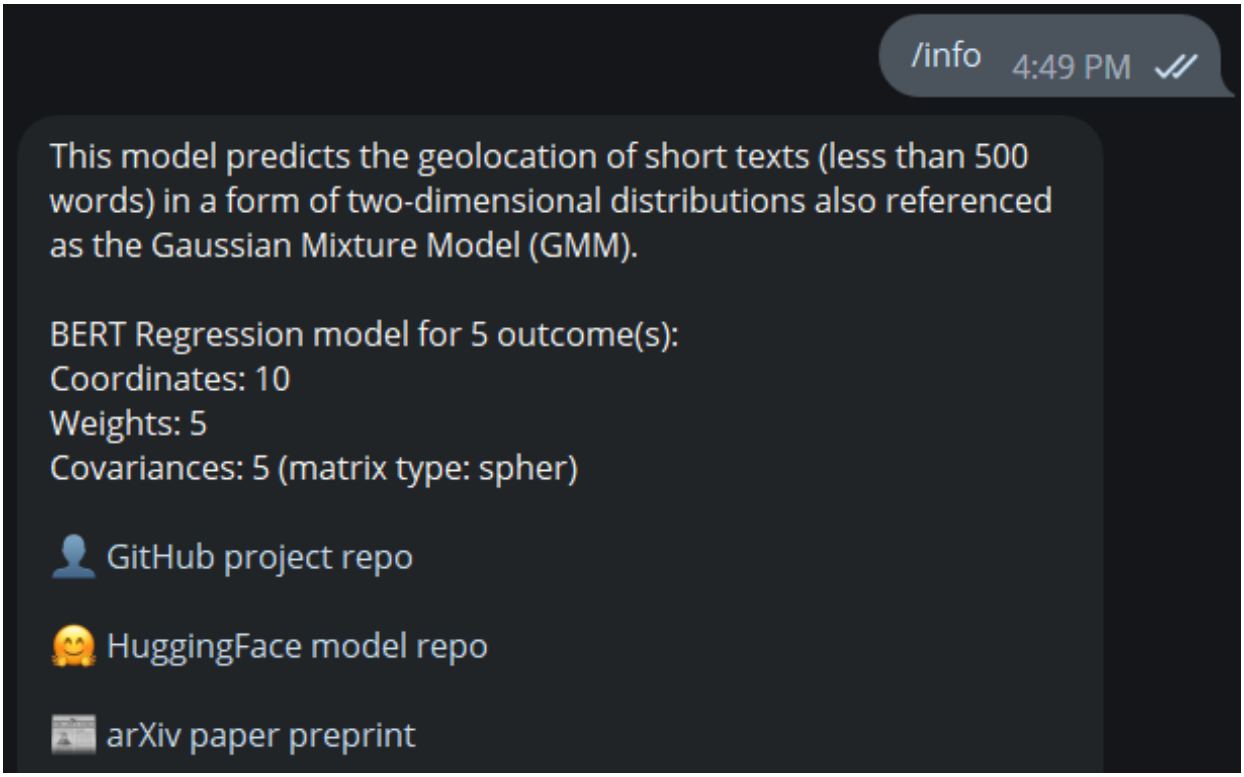


Рисунок 3.2 – Відповідь бота на команду “/info”

/predict <text> – отримати результати прогнозування у текстовому вигляді та у вигляді графіків GMM як спектру PDF двовимірних розподілів на мапі Меркатора як показано на Рисунку 3.3

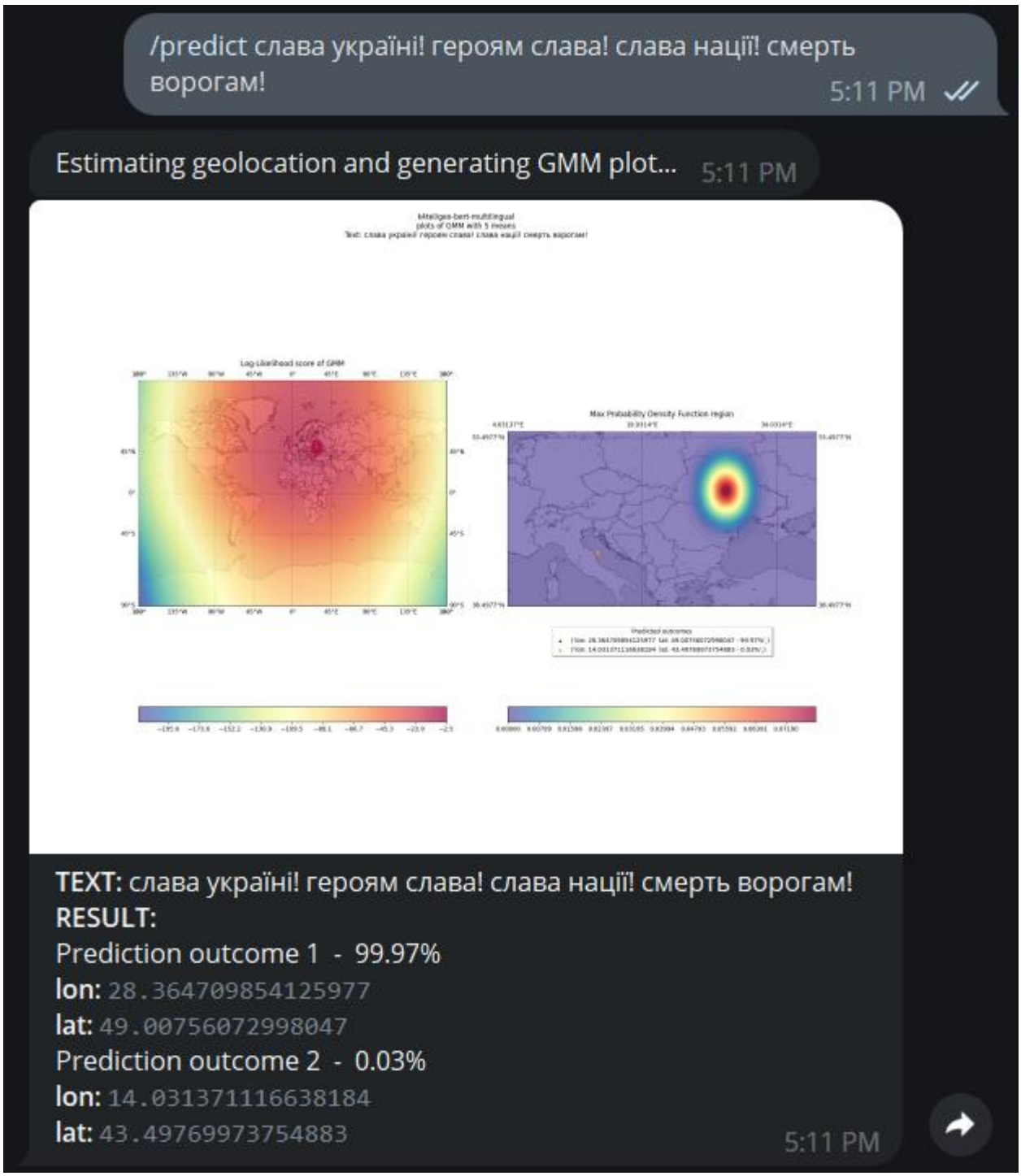


Рисунок 3.3 – Відповідь бота на команду “/predict”

Для Розробника – гілка predict – ПЗ для прогнозування місцезнаходження за текстом з консолі використовуючи готову модель на HuggingFace

Для запуску передбачення одного тексту завантажується модель репозиторію HF. Для запуску передбачення з налаштуваннями за замовчуванням виконайте:

```
& python text_result.py
```

Налаштування слід змінювати вручну, змінюючи вміст функції main() у скрипті. Існують варіанти використання спеціального HF-пайплайну або ручної обробки входів і виходів.

Для Розробника – ПЗ для навчання та оцінки власних моделей на власних датасетах

Щоб запустити навчання з точним налаштуванням, помістіть файл набору даних (.jsonl), що містить колонки "lon", "lat", "text", "user" і "place" (поля JSON, заголовки не потрібні), до теки **datasets**.

Потім змініть ім'я файлу набору даних у *train_bert.py* вручну або за допомогою аргументу -d <ім'я_файлу_набору_даних>.jsonl.

Навчання:

Для запуску тренування з налаштуванням гіперпараметрів за замовчуванням запустіть:

```
& python train_bert.py --train
```

Ви можете змінити гіперпараметри за замовчуванням вручну в *train_bert.py* або передати аргументи командного рядка за допомогою попередньо визначених прапорів. Список усіх прапорів можна знайти у тому ж файлі точки входу. На практиці швидкість навчання, тип планувальника, кількість епох, параметри функції втрат та цільові стовпці мають залишатися незмінними.

Параметри, які зазвичай змінюються, включають кількість результатів, тип коваріації, функції, ім'я файлу набору даних, розмір навчального завантажувача даних, розмір партії та крок журналу.

Під час налаштування моделей метрики навчання та тестові метрики (обчислюються в кінці кожної епохи) записуються до папки runs. Відстеження продуктивності моделей реалізовано за допомогою python-бібліотеки Tensorboard. Файли моделей та їх контрольних точок автоматично зберігаються в папку models.

Оцінювання:

Щоб запустити оцінювання, помістіть файл набору даних у папку datasets і переконайтеся, що у вас є файл налаштованої моделі у форматі .pth у каталозі models.

Для запуску оцінки з налаштуваннями за замовчуванням виконайте

```
& python train_bert.py --eval
```

У цьому випадку файл моделі буде обрано автоматично відповідно до префікса імені файлу, сформованого із заданих гіперпараметрів. Щоб вибрати модель вручну, вам слід налаштувати гіперпараметри (кількість результатів, тип коваріації, особливості, тип функції втрат) відповідно до попередньо налаштованої моделі та запустити її:

```
& python train_bert.py --eval -m <назва_моделі>
```

Типовими параметрами для оцінки є ім'я файлу набору даних, розмір валідаційного завантажувача даних та ім'я файлу моделі.

Для виконання оцінки для кожного користувача використовуйте прапори -vu -v <N>, які виберуть N користувачів з найбільшою кількістю вибірок з набору даних. У цьому випадку для обчислення метрик продуктивності беруться середні значення на користувача, а не середні значення на твіт. Зауважте, що тільки ймовірнісні моделі, що використовують GMM, можуть підсумовувати кілька прогнозів на твіт.

Результати оцінювання записуються у файл набору даних .jsonl, що містить вхідні та вихідні дані моделі. За замовчуванням, показники ефективності обчислюються в кінці і записуються в короткий файл звіту у форматі .txt. Візуалізація щільності відстані помилки та її кумулятивного розподілу за результатами виводиться у файли формату .png. За допомогою valid_map.py ви можете читати збережені файли прогнозів і легше

використовувати функції візуалізації. Всі результати оцінювання зберігаються в папці results.

Прогнозування:

ПРИМІТКА!!! Для запуску прогнозу з одним текстом ви повинні помістити файли моделі у форматі .pth до каталогу models/final.

Для запуску передбачення з налаштуваннями за замовчуванням запустіть:

```
& python input_entry.py
```

Параметри, такі як кількість результатів, імовірнісний або геопросторовий тип моделі, локальний файл моделі та текст можуть бути вказані за допомогою прапорів:

```
& python input_entry.py -m <ім'я_файлу_моделі> -t <текст>
```

Додатково, за підтримкою можна писати на lutsai.k@gmail.com

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Алгоритм та програмне забезпечення для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT

Графічний матеріал

КПІ.ІТ-9116.045490.06.99

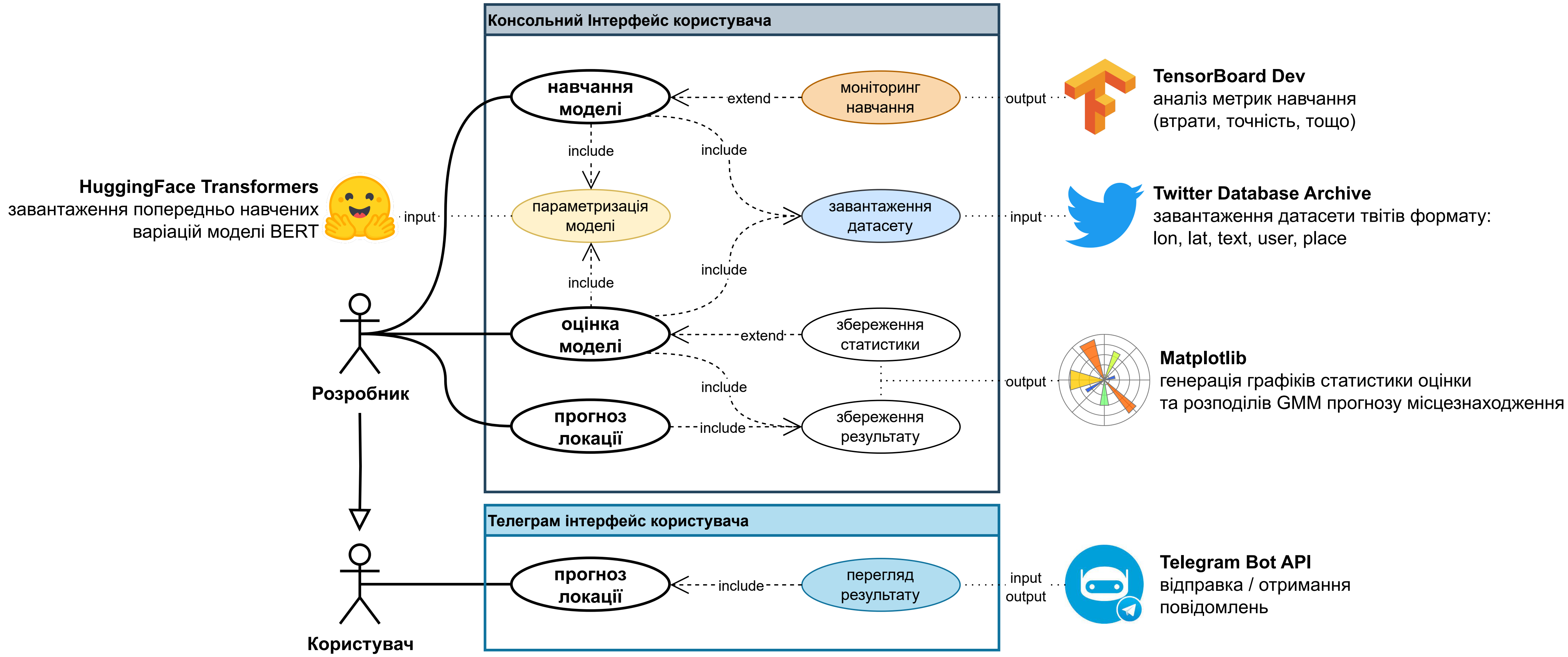
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ігор БАКЛАН

Нормоконтроль:
_____Максим ГОЛОВЧЕНКО

Виконавець:
_____ Катерина ЛУЦАЙ

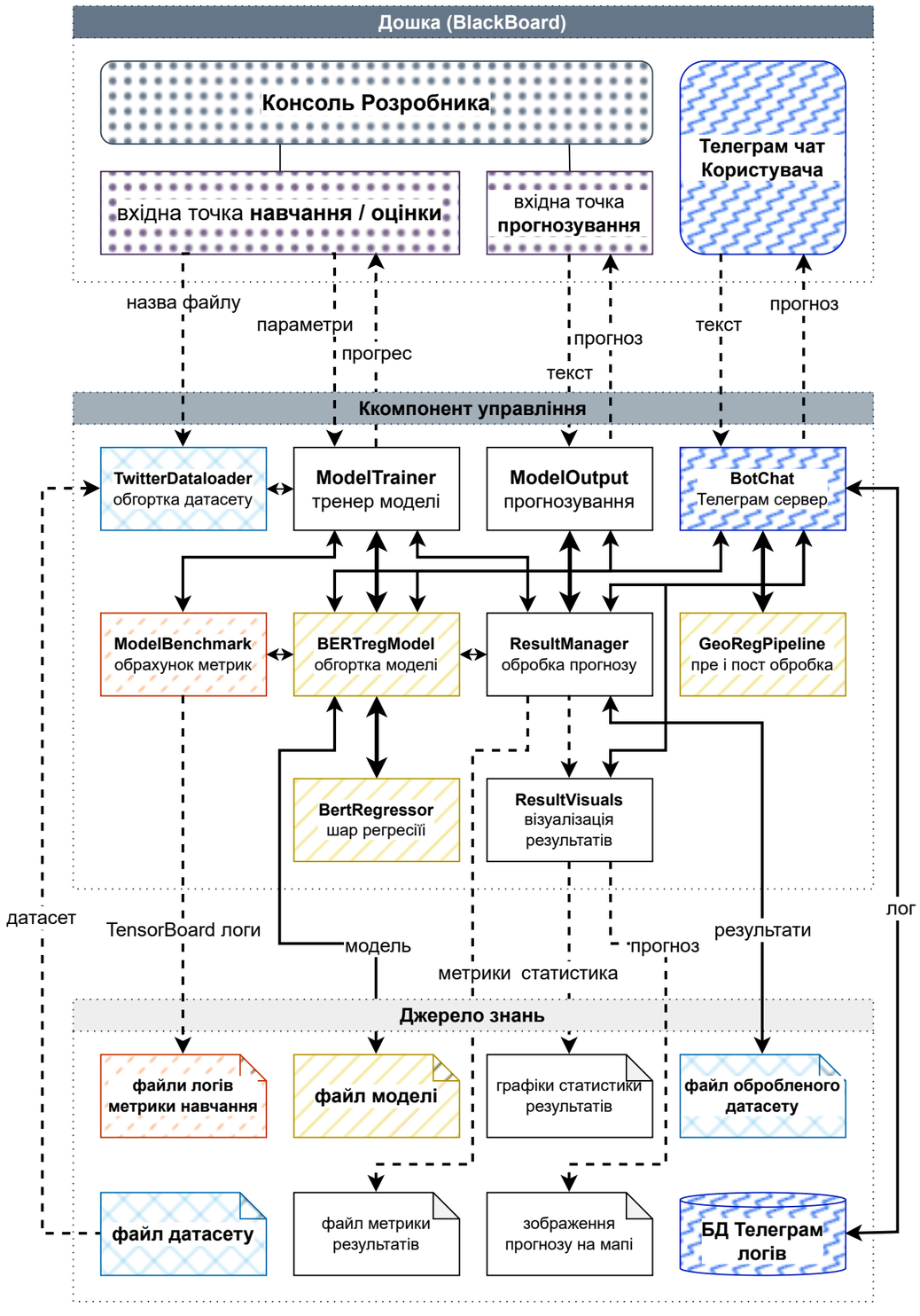
Київ – 2023

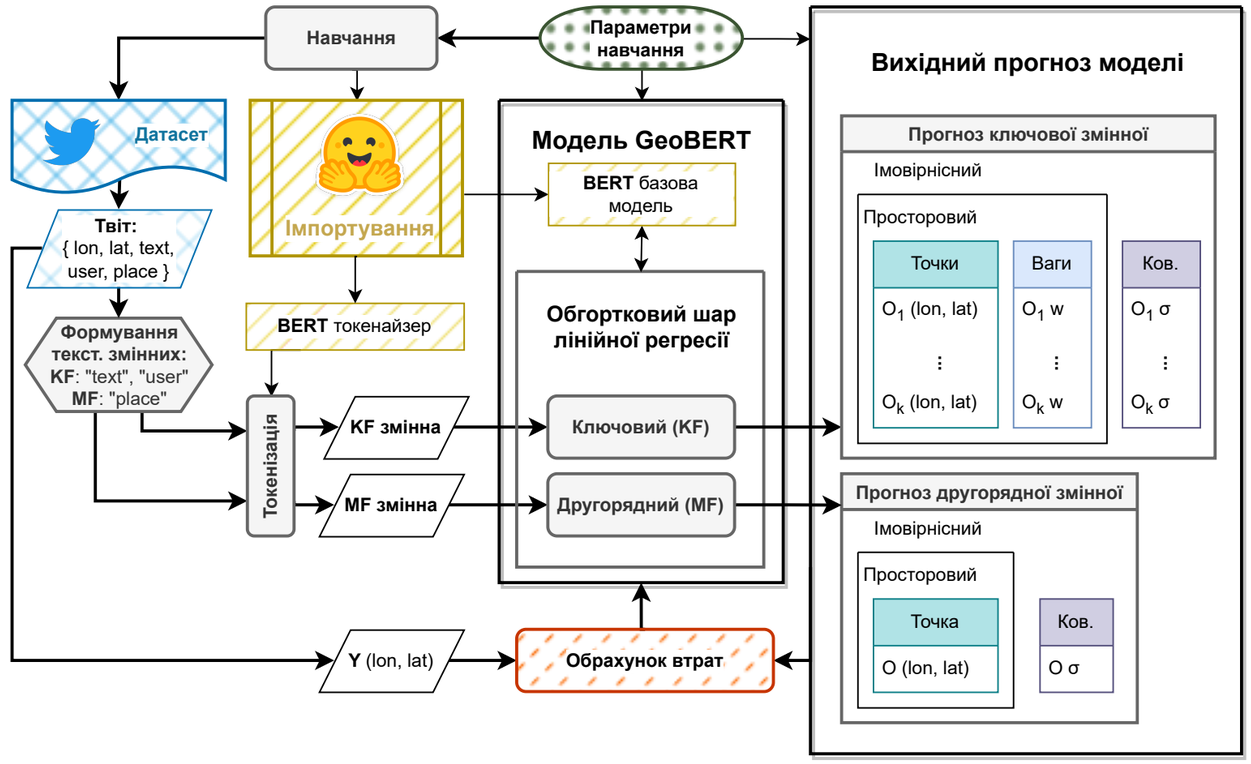
КПІ.ІТ-9116.045490.02.81.ССВВ



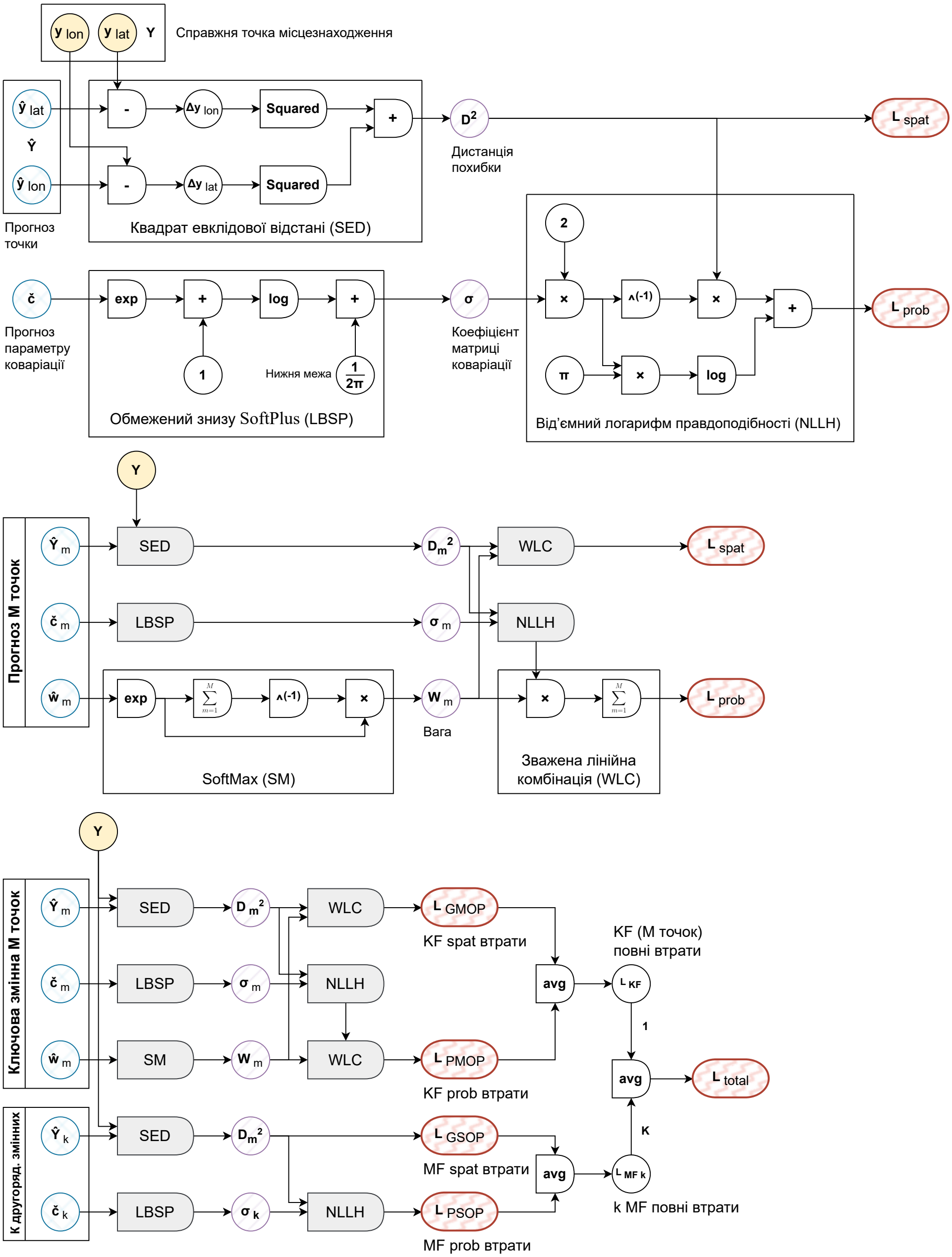
					КПІ.ІТ-9116.045490.06.99.ССВВ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використання	Літера / Маса / Масштаб
Розробив		Луцай К. А.				
Перевірив		Баклан І. В.				Аркуш / Аркушів
Т. кон.					Алгоритм та ПЗ для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91
Н. кон.		Головченко М.М.				
Затвердив		Жаріков Е. В.				

КПІ.ІТ-9116.045490.06.99.ССМ





					КПІ.ІТ-9116.045490.06.99.ССМ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера: — Маса: — Масштаб: —
Розробив		Луцай К. А.				Аркуш: — Аркушів: —
Перевірив		Баклан І. В.				
Т. кон.						
Н. кон.		Головченко М.М.			Алгоритм та ПЗ для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91
Затвердив		Жаріков Е. В.				



					КПІ.ІТ-9116.045490.06.99.ССД
Зм.	Арк.	№ докум.	Підп.	Дата	
Розроб.		Луцай К. А.			Схема структурна діяльності
Перев.		Баклан І. В.			Лит. / Маса / Масштаб
Т. Кон.					Аркуш / Аркушів
Н. Кон.		Головченко М.М.			Алгоритм та ПЗ для прогнозування геолокації у соціальних мережах за допомогою моделей на основі BERT
Затв.		Жаріков Е. В.			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91