НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“___”_____2024 р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Система переднавчання великомовних моделей на основі даних користувача

Виконав: студент 4 курсу, групи ІО-05
(шифр групи)

Басенко Нікіта Романович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ас. Русінов В.В _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормконтроль) ас. Ковальчук О. М _____
(назва розділу) (посада; вчене звання; науковий ступінь; прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2024 рік

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“____”_____2024 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Басенка Нікіти Романовича

1. Тема проєкту *Система переднавчання великомовних моделей на основі даних користувача*

керівник проєкту ас. Русінов В.В.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від _____.

2. Термін здачі студентом закінченого проєкту *3 червня 2024 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Опис великомовних моделей та аналіз існуючих рішень.

Розділ 2. Огляд технологій дотренерування моделей

Розділ 3. Реалізація Проекту

Розділ 4. Перевірка та Аналіз Роботи

Київ – 2024 рік

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) схема роботи алгоритму підготовкки даних, функціональна схема (діаграма класів) і алгоритм дій програмного забезпечення .

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	ас. Ковальчук О. М		

7. Дата видачі завдання

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	10.12.2023-15.12.2023	
2.	*Вивчення та аналіз завдання*	15.12.2023-15.-.3.2024	
3.	*Розробка архітектури та загальної структури системи*	15.03.2024-25.03.2024	
4.	*Розробка структур окремих підсистем*	25.03.2024-5.04.2024	
5.	*Програмна реалізація системи*	5.04.2024-15.04.2024	
6.	*Оформлення пояснювальної записки*	13.05.2024-30.05.2024	
7.	*Захист програмного продукту*		
8.	*Передзахист*		
9.	*Захист*		

Студент-дипломник _____ Басенко Нікіта
(підпис)

Керівник проєкту _____ Русінов Володимир
(підпис)

АНОТАЦІЯ

Цей дипломний проєкт присвячено дотренуванню великомовних моделей на основі спеціалізованих користувацьких даних. Метою проєкту є адаптація мовної моделі для генерації тексту специфічної тематики, що дозволить покращити її продуктивність та точність. В рамках роботи були вивчені підходи до дотренування моделей, такі як LoRa.

Під час виконання проекту були зібрані тематичні данні з соціальних мереж які слугували основним корпусом для дотренування.

Ключові слова: великомовні моделі, дотренування, LoRa, transformers

ANNOTATION

This thesis project focuses on training large-scale language models based on specialized user data. The goal of the project is to adapt a language model to generate text on specific topics, which will improve its performance and accuracy. As part of the work, approaches to model training, such as LoRa, were studied.

During the project, thematic data from social networks were collected and used as the main corpus for training.

Keywords: large language models, fine-tuning, LoRa, transformers

Справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система переднавчання	3		
			Великомовних моделей на			
			основі даних користувача			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система переднавчання	47		
			Великомовних моделей на			
			основі даних користувача			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система переднавчання	1		
			Великомовних моделей на			
			основі даних користувача			
			Схема роботи алгоритму			
			Підготовки даних			
	А4	***ІАЛЦ.467200.005 Д2***	Система переднавчання	1		
			Великомовних моделей на			
			основі даних користувача			
			Функціональна схема			
			(діаграма класів)			
	А4	***ІАЛЦ.467200.006 Д3***	Система переднавчання	1		
			Великомовних моделей на			
			основі даних користувача			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Система переднавчання	11		
			Великомовних моделей на			
			основі даних користувача			
			Текст програмного коду			

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	

ІАЛЦ.467200.001 ОА

Розроб		Басенко Н. Р.			*Система переднавчання великомовної моделі на основі даних кристувача* Опис альбому
Перев		Русінов В.В			

Літ.	Аркуш	Аркушів
	1	1

“КПІ” ім. Ігоря Сікорського ФІОТ ІО-05

Київ – 2024 рік

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система переднавчання великомовних моделей на основі даних користувача*»

Київ – 2024 рік

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання поширюється на розробку системи дотренування великомовних моделей.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр Комп'ютерні системи та мережі», який був затверджене кафедрою Обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою проекту є розробка системи дотренування великомовних моделей.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерела, що були використані для розробки є науково-технічна література яка стосується великомовних моделей, їх структури, архітектури та методів тренування.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблений метод має виконувати такі вимоги:

- Генерація тексту за запитом користувача.
- Проста та зрозуміла документація, для послідуючого використання алгоритму.

5.2. Вимоги до програмного забезпечення

- ОС Windows, MacOS або будь-який дистрибутив Linux
- Підтримка Python 3.10+
- Наявний великий датасет тренувальних даних

5.3. Вимоги до апаратної частини

- Наявний GPU або TPU
- 16GB і більше оперативної пам’яті

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2023-15.12.2023
Вивчення та аналіз завдання	15.12.2023-15.03.2024
Розробка архітектури та загальної структури системи	15.03.2024-25.03.2024
Розробка структур окремих частин системи	25.03.2024-5.04.2024
Програмна реалізація системи	05.04.2024-15.04.2024
Виправлення помилок	15.04.2024-20.05.2024
Оформлення пояснювальної записки	13.05.2024-30.05.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система переднавчання великомовної моделі на основі данних копистувача*»

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

LLM	Large Language Model.
GPT	Generative Pre-trained Transformer
LSTM	Long short-term memory.
NLP	Natural Language Processing.
RNN	Recurrent Neural Networks.

ВСТУП

За останні 2 роки людство охопив бум штучного інтелекту. Починаючи від звичних систем рекомендацій контенту та перевірки його на приналежність до специфічної тематики в соціальних мережах до простого додатку по вимірюванню пульсу. Через це різні компаніїї почали цікавитися цим напрямком так як захотіли привабити до своїх продуктів нову аудиторію, впроваджуючи цей самий штучний інтелект до своїх сервісів. Таким чином з'явилися особисті помічники чи чат боти, які можуть не тільки відповісти на ваше питання за допомогою інформації на сайті а й дати якусь рекомендіцію чи просто поговорити з вами на ту чи іншу тему.

На сьогоднішній день більшість сервісів використовують платне API від компанії OpenAI. Але через проблему масштабованності, приватності та неточності у спеціалізованих задачах, подібні методи не являються пріорітетними у європейських компаніях. Саме тому існують спеціалізовані рішення, які орієнтовані саме на конкретну діяльність компаній. Але саме через це вони можуть коштувати великих грошей і часу.

Саме це послужило ідеєю розробки спеціального рішення, яке здатне підлаштувати існуючі великомовні моделі машинного навчання до спеціалізованих задач бізнесу та не тільки. Подібне рішення здатне забезпечити безпеку витіку даних до великих корпорацій, та зменшити поріг доступу до повноцінних рішень, таких як GPT, Cloud та інші.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дотренування великих мовних моделей

1.1.1 Вступ до великомовних моделей

Великомовні моделі (LLM) представляють собою одне з найважливіших досягнень дослідників у галузі обробки та розуміння природньої мови (NLP) що є підмножиною машинного навчання (ML). Подібні моделі, такі як GPT, розроблені для розуміння та генерації людського тексту на високому рівні. Історія великих мовних моделей починається з базових концепцій машинного навчання та нейронних мереж, які були застосовані для обробки текстових даних.

1.1.2 Історія та розвиток LLM

Ранні моделі для обробки природної мови, такі як N-грами та моделі на основі сумки слів, мали обмежені можливості в розумінні контексту та генерації зв'язного тексту. З розвитком нейронних мереж, зокрема рекурентних нейронних мереж (RNN) та моделей з довготривалою короткочасною пам'яттю (LSTM), з'явилися більш потужні підходи до обробки послідовностей тексту. [1] Проте, справжній прорив стався з винаходом архітектури «Трансформера», яка була запропонована у статті "Attention is All You Need" у 2017 році розробниками з Гугл. [2]

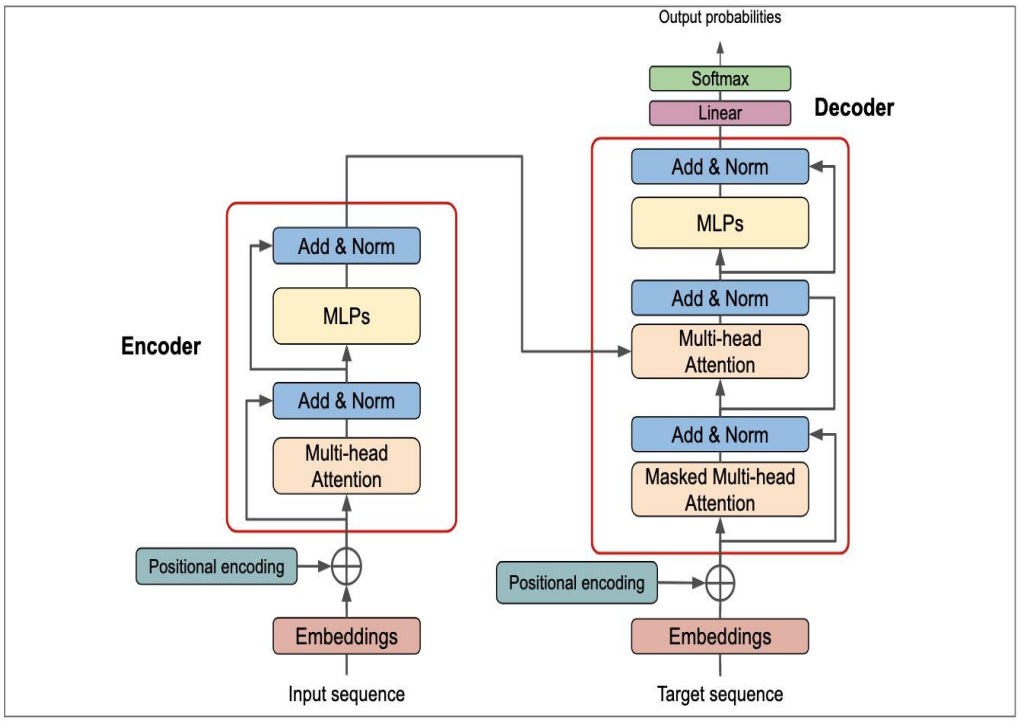


Рисунок 1.1. – Архітектура трансформерів

Трансформери, зокрема широковідомі нині моделі GPT такі як GPT4 та ChatGPT, дозволили значно покращити якість генерації тексту та розуміння контексту завдяки використанню механізму уваги (attention mechanism), який ефективно обробляє залежності між словами на різних відстанях.

1.1.3 Призначення великих мовних моделей

Великі мовні моделі призначені для широкого спектру завдань, включаючи, але не обмежуючись:

1.Генерація тексту: створення зв'язного тексту на основі заданого початкового фрагмента або теми;

2.Автоматичний переклад: переклад тексту з однієї мови на іншу з високою точністю

3.Аналіз настроїв: визначення емоційного тону тексту

4.Питання-відповідь системи: надання відповідей на запитання на основі великого корпусу текстових даних

5.Резюмування тексту: створення короткого викладу довгих текстових документів

6.Розпізнавання іменованих сутностей (NER): ідентифікація ключових іменованих об'єктів у тексті, таких як імена людей, місця, організації тощо.

Більшість з цих задач є широковідомими навіть серед звичайних нейромереж специфічного направлення, але всі вони не мають всіх задач в одній собі. Що робить подібну архітектуру унікальною та незамінною. [3]

1.1.4 Необхідність дотренування моделей

Незважаючи на велику універсальність великомовних моделей, вони часно тренуються на величезних загальних корпусах даних, таких як Common Crawl, Wikipedia та інші великі тексти з мережі Інтернет. В додаток через їх великий розмір, який часто доходить до террабайтів, мало хто підходить до задачі відбору даних з розумом. Ці дані охоплюють широкий спект тем, але не можуть враховувати специфіку окремих галузей або вузькоспеціалізованих задач. Тут і вступає в гру дотренування моделей.

Дотренування великих мовних моделей полягає в адаптації вже навчених моделей до конкретних користувацьких даних. Це включає додаткове тренування моделі на новому наборі даних, який є більш релевантним для конкретного завдання або галузі.

1.1.5 Причини важливості дотренування

1.Спеціалізація моделі: дотренування дозволяє моделі краще розуміти специфічні терміни та контексти, що є важливими для конкретної галузі. Наприклад, модель, дотренована на медичних даних, буде краще обробляти текст, пов'язаний з медициною, або ж краще відповідати на часті запитання якоїсь фірми.

2.Покращення точності: адаптація або коригування моделі до специфічних даних допомагає збільшити точність і правильність відповідей. Це особливо важливо для задач, де помилки можуть мати серйозні наслідки, наприклад, в медичній або консультаційній сферах.

3.Зменшення обчислювальних витрат: дотренування на обмеженому наборі даних може бути менш обчислювально затратним, ніж тренування нової моделі з нуля. Це дозволяє економити ресурси та швидше досягати потрібних результатів.

4.Актуальність даних: використання нових, актуальних даних для дотренування дозволяє моделі залишатися в курсі останніх змін у певній галузі або темі. Це особливо важливо в умовах швидкозмінних інформаційних середовищ, таких як соціальні мережі або наукова діяльність. [4]

1.2 Типові задачі великомовних моделей

Великі мовні моделі (LLM) здатні вирішувати широкий спектр завдань завдяки їхньому глибокому розумінню мови та контексту. У цьому підрозділі ми розглянемо основні типові задачі, які виконують великі мовні моделі, та специфіку підготовки даних для кожного з них. Основні задачі включають резюмування (summarization), переклад (translation), питання-відповідь (question answering), генерацію тексту (text generation) та інші.

1.2.1 Резюмування

Резюмування тексту є завданням створення короткого викладу довгого тексту, зберігаючи ключову інформацію. Для тренування моделей на цю задачу використовуються спеціальні датасети, що складаються з пар початкового тексту і його резюме.

Процес тренування: модель навчається на парах текстів, порівнюючи згенероване резюме з бажаним результатом за допомогою метрик, таких як ROUGE (Recall-Oriented Understudy for Gisting Evaluation).

1.2.2 Переклад

Переклад тексту з однієї мови на іншу є однією з найпоширеніших задач для великих мовних моделей. Для цієї задачі використовуються паралельні корпуси, які містять тексти на вихідній та цільовій мовах.

Процес тренування: модель навчається відповідати тексту оригіналу текстом перекладу, використовуючи метрики якості перекладу, такі як BLEU (Bilingual Evaluation Understudy).

1.2.3 Відповіді на питання

Завдання питання-відповідь передбачає надання відповідей на запитання на основі додаткового контексту. Для тренування використовуються датасети, що містять пари запитань та відповідних контекстів. Частіше за все моделі на вхід подається питання користувача, контекст, з якого модель повинна отримати відповідь а в якості відповіді моделі очікується коротке повідомлення, текст з якого взятий з цього доданого контексту.

1.2.4 Генерація тексту

Генерація тексту є базовою задачею для великомовних моделей, де модель створює новий текст на основі вхідного контексту або теми. Для цього використовуються датасети, що складаються з текстових фрагментів, які модель навчається продовжувати. [5].

LLM також використовуються для інших задач, таких як (NER), аналіз настроїв, класифікація тексту та інші, які орієнтовані на класичні задачі

машинного навчання, але для великомовних моделей вони використовуються у дуже специфічних задачах, через що ми не будемо брати їх до уваги [6].

1.3 Огляд існуючих рішень

В цьому розділі ми розглянемо чотири відомі приклади дотренованих великих мовних моделей (LLM), які продемонстрували високі результати в своїх спеціалізованих задачах. Ми детально розглянемо MedPaLM, CodeLlama, а також ще дві відомі моделі: BioBERT та DIALoGPT. Для кожної з цих моделей ми проаналізуємо основу, на якій вони були створені, цілі, задачі, які вони виконують, а також методи дотренування та особливості.

1.3.1 MedPaLM

MedPaLM - це дотренована модель для медичної інформатики, розроблена на основі архітектури GPT (Generative Pre-trained Transformer). Вона була створена командою дослідників з Google Health та DeepMind для покращення взаємодії в медичному середовищі. MedPaLM базується на моделі PaLM, яка є власної розробкою Google, та є закритою моделлю як і майже бідь яка від цієї компанії, яка не підпадає до категорії відкритого коду та ваг, через що повторити її або використати в якості основи стає неможливим. Питання відкритості даних та будь яких досліджень взагалі стає відкритою темою адже це одне з найконсервативніших сфер, де майже за кожне дослідження тобі необхідно платити. Метою MedPaLM є створення інструменту для медичних фахівців, який може допомагати у діагностиці, наданні рекомендацій та відповіді на медичні запити. Модель використовується для різноманітних медичних задач, таких як автоматичне резюмування медичних статей, відповіді на запитання пацієнтів та лікарів, генерація рекомендацій на основі симптомів. MedPaLM була дотренована на великих медичних корпусах,

включаючи медичні статті, записи пацієнтів та діагностичні рекомендації. Використовувалися методи контролю якості, такі як валідація експертами-медиками, щоб забезпечити точність і надійність моделей. Ця модель, через специфіку своєї сфери не може бути оцінена алгоритмами так як на пряму може впливати на людське життя і на людський фактор, через що дотреновувалась лише за допомогою коментарів та зворотнього зв'язку справжніх фахівців та незалежної групи експертів.

1.3.2 CodeLlama

CodeLlama - це спеціалізована мовна модель для задач програмування, розроблена на основі моделі LLaMa. Вона була створена командою Meta для покращення процесу написання та розуміння коду. Модель може генерувати фрагменти коду, надавати рекомендації з виправлення помилок, допомагати в написанні документації та коментарів до коду. CodeLlama була дотренована на великому обсязі даних з відкритих репозиторіїв коду, у партнерстві з GitHub, включаючи приклади коду на різних мовах програмування. Використовувалися спеціальні алгоритми та підходи для розпізнавання структурних елементів програмування, коду та їхнього контексту

1.3.3 BioBERT

BioBERT - це спеціалізована модель для біомедичних текстів, розроблена на основі моделі BERT (Bidirectional Encoder Representations from Transformers). Вона була створена командою дослідників з Кореї для покращення аналізу біомедичних текстів. BioBERT базується на моделі BERT, яка забезпечує високий рівень розуміння контексту завдяки двонаправленому тренуванню. Основною метою BioBERT є покращення точності та ефективності аналізу біомедичних та наукових текстів, таких як наукові статті та клінічні записи.. BioBERT була дотренована на великому обсязі

біомедичних текстів, включаючи PubMed та PMC статті. Було використано контрольоване навчання, за схожою методикою з MedPaLM , для покращення точності розпізнавання біомедичних термінів та понять.

1.3.4 DIALoGPT

DIALoGPT - це розширена та дотренована версія моделі GPT-2, спеціально адаптована для ведення діалогів з реальними людьми. Вона була розроблена командою Microsoft Research для покращення взаємодії з користувачами у форматі діалогів для внутрішніх рішень компанії, таких як чатботи та команда підтримки на сайтах. Модель використовується для ведення діалогів з користувачами, надання відповідей на запити та створення релевантного контексту для розмови. DIALoGPT була дотренована на великому обсязі діалогових даних з публічних форумів та соціальних мереж, окрім того мала доступ до частих питань-відповідей та діалогів компанії. Використовувалися методи контролю якості, такі як людські оцінки та зворотний зв'язок, для покращення релевантності та природності відповідей.

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі я представив ґрунтовний огляд великих мовних моделей (LLM), їхньої історії, призначення та типових задач. Детально пояснив необхідність дотренування цих моделей для адаптації до специфічних галузей та завдань. Ключовими перевагами дотренування є підвищення точності, спеціалізація моделі до певної предметної області, зменшення обчислювальних витрат та актуалізація даних.

Я описав основні типові задачі LLM, такі як резюмування, переклад, питання-відповідь та генерацію тексту. Для кожної задачі навів детальну інформацію щодо специфіки підготовки даних та процесу тренування моделей.

Крім того, я проаналізував чотири відомі приклади успішно дотренованих LLM: MedPaLM для медичної галузі, CodeLlama для програмування, BioBERT для біомедичних текстів та DIALoGPT для ведення діалогів. Розглянув базові моделі, цілі, задачі, датасети та методи дотренування для кожного з цих прикладів.

Аналіз відомих рішень та специфіка роботи самої великомовної моделі показали, що дотренування під конкретну задачу чи сферу є дуже хорошим рішенням для всіх сторін. Воно здатне покращити роботу систем, де ця модель буде використовуватись, спростити та покращити комунікацію з людьми, які будуть взаємодіяти з моделлю. Це значно полегшить розробку нових рішень, надасть кращий доступ до матеріалів та може залучити додаткові інвестиції.

Загалом, цей розділ закладає міцний фундамент для розуміння великих мовних моделей, їхніх можливостей, обмежень та необхідності адаптації шляхом дотренування до конкретних вимог.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ

2.1 Архітектура трансформера

Архітектура трансформера є основою сучасних великих мовних моделей, таких як GPT (Generative Pre-trained Transformer). Вона була вперше представлена у статті "Attention is All You Need" (2017) і з того часу стала ключовою технологією в галузі обробки природної мови (NLP). У цьому підрозділі ми детально розглянемо архітектуру трансформера, її основні компоненти, такі як механізм уваги (attention mechanism), енкодер та декодер, та їхню взаємодію.

2.1.1 Основні компоненти трансформера

Механізм уваги є центральним елементом трансформера, який дозволяє моделі зосереджуватися на різних частинах вхідної послідовності під час обробки кожного слова. Існує кілька типів уваги, але найпоширенішим є багатоголовий механізм уваги (multi-head attention), теми які ми розглянемо далі, наведені на рис.1.1 [2]

- Багатоголовий механізм уваги (Multi-Head Attention):

1.Вхідні дані: Вхідними даними для механізму уваги є трійки матриць: запити (queries), ключі (keys) та значення (values).

2.Процес: Запити, ключі та значення проектуються в декілька просторів меншої розмірності за допомогою лінійних перетворень. Кожна така проекція називається "головою".

3.Обчислення ваг: Для кожної голови обчислюються ваги уваги шляхом множення запитів на ключі та нормалізації результату за допомогою softmax.

4.Зважені значення: Отримані ваги використовуються для зважування значень, що дозволяє моделі "уважніше" дивитися на різні частини вхідної послідовності.

5.Конкатенація та проекція: Зважені значення з усіх голів конкатенуються і проектуються назад у початковий простір розмірностей за допомогою лінійного шару.

- Енкодер (Encoder)

Енкодер відповідає за обробку вхідної послідовності та створення контекстно-залежних представлень кожного елемента послідовності. Він складається з кількох однакових блоків (layers), кожен з яких містить два основні компоненти:

1.складається з кількох однакових блоків (layers), кожен з яких містить два основні компоненти

2.Позиційно-залежний зворотний зв'язок (Position-wise Feed-Forward Network): Двошаровий перцептрон, який застосовується до кожного елемента послідовності незалежно.

Далі наведено алгоритм обробки вхідної послідовності:

1.Вхідні ембеддінги: Спочатку вхідна послідовність перетворюється на ембеддінги (векторні представлення) за допомогою вбудованого шару (embedding layer)

2.Вхідні ембеддінги: Спочатку вхідна послідовність перетворюється на ембеддінги (векторні представлення) за допомогою вбудованого шару (embedding layer)

3.Процес обробки: Ембеддінги проходять через кожен блок енкодера, де вони обробляються багатоголовим механізмом уваги та позиційно-залежним зворотним зв'язком

- Декодер(Decoder)

Декодер відповідає за генерацію вихідної послідовності на основі контексту, створеного енкодером. Декодер також складається з кількох однакових блоків, кожен з яких містить три основні компоненти:

1.Багатоголовий механізм уваги: Перший механізм уваги в кожному блоці декодера обробляє попередні слова вихідної послідовності.

2.Багатоголовий механізм уваги з перехресною увагою: Другий механізм уваги обробляє вихідну послідовність у контексті вхідної послідовності, використовуючи представлення, створені енкодером.

3.Позиційно-залежний зворотний зв'язок: Як і в енкодері, застосовується двошаровий перцептрон для кожного елемента послідовності.

Далі наведено алгоритм обробки вихідної послідовності:

1.Вхідні ембеддінги: Подібно до енкодера, вихідна послідовність перетворюється на ембеддінги.

2.Позиційні ембеддінги: Додаються позиційні ембеддінги для збереження інформації про порядок слів.

3.Процес обробки: Ембеддінги проходять через кожен блок декодера, де вони обробляються механізмом уваги, механізмом перехресної уваги та позиційно-залежним зворотним зв'язком.

- Взаємодія енкодера та декодера

Взаємодія між енкодером та декодером є ключовим елементом роботи трансформера. Енкодер створює контекстні представлення вхідної послідовності, які використовуються декодером для генерації вихідної послідовності.

Далі наведено приклад взаємодії енкодера та декодера:

Розглянемо простий приклад перекладу речення з англійської на українську мову.

1.Вхідне речення: "The cat is on the mat."

2.Енкодер:

-Вхідне речення перетворюється на ембеддінги.

-Додаються позиційні ембеддінги.

-Ембеддінги проходять через кілька блоків енкодера, де обробляються механізмом уваги та позиційно-залежним зворотним зв'язком.

-Енкодер створює контекстні представлення для кожного слова в реченні.

3.Декодер:

-Вихідна послідовність (наприклад, початок перекладу "Кіт") перетворюється на ембеддінги.

-Додаються позиційні ембеддінги.

-Ембеддінги проходять через кілька блоків декодера, де обробляються механізмом уваги та перехресної уваги, використовуючи контекстні представлення з енкодера.

-Декодер генерує наступне слово у вихідній послідовності, наприклад, "на".

Цей процес повторюється до завершення перекладу ("Кіт на килимку.").

2.2 Традиційні методи генерації тексту

У цьому підрозділі ми порівняємо архітектуру трансформера з традиційними методами генерації тексту, такими як LSTM (Long Short-Term Memory), RNN (Recurrent Neural Networks) та GAN (Generative Adversarial

Networks). Кожен з цих методів має свої принципи роботи, переваги та недоліки, які ми детально розглянемо з технічної точки зору. [6].

2.2.1 LSTM

LSTM є покращенною версією рекурентних нейронних мереж (RNN), спеціально розроблених для подолання проблеми зникання градієнта, що часто виникає у традиційних RNN. Основними компонентами LSTM є клітини пам'яті та три типи гейтів або вентилів: вхідний гейт, вихідний гейт і гейт забуття[7]

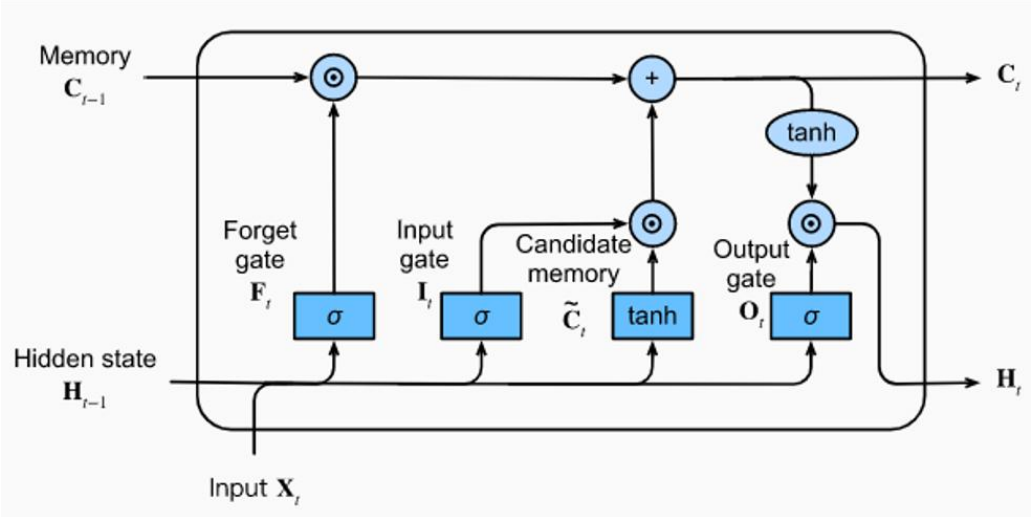


Рисунок 2.1 – Архітектура LSTM

- Клітини пам'яті: Зберігають інформацію протягом довгих періодів часу

- Вхідний гейт: Контролює, яку частину нової інформації слід зберегти в клітині пам'яті.

- Гейт забуття: Вирішує, яку частину старої інформації слід забути.

- Вихідний гейт: Визначає, яку частину інформації з клітини пам'яті слід використовувати як вихід.

- Вихідний гейт: Визначає, яку частину інформації з клітини пам'яті слід використовувати як вихід.

- Переваги:

- Збереження довгострокової залежності: LSTM ефективно зберігає інформацію протягом довгих послідовностей, що є корисним для задач генерації тексту, що бореться із проблемою градієнту
- Можливість обробки послідовних даних: LSTM добре підходить для роботи з часовими рядами та послідовностями, де порядок даних є важливим. Саме тому їх часто використовують у задачах зі даними у часі.

- Недоліки

- Висока обчислювальна складність: Через складну архітектуру LSTM потребує більше обчислювальних ресурсів.

- Проблеми з масштабуванням: LSTM важко масштабувати для дуже великих послідовностей через високі вимоги до пам'яті та все ту ж проблему зникаючого градієнту.

2.2.2 RNN

RNN є попередником LSTM і використовуються для обробки послідовних даних шляхом циклічного зв'язку між нейронами. У традиційних RNN кожен вихід попереднього кроку подається як вхід для наступного кроку-застосунку разом із первинним входом.

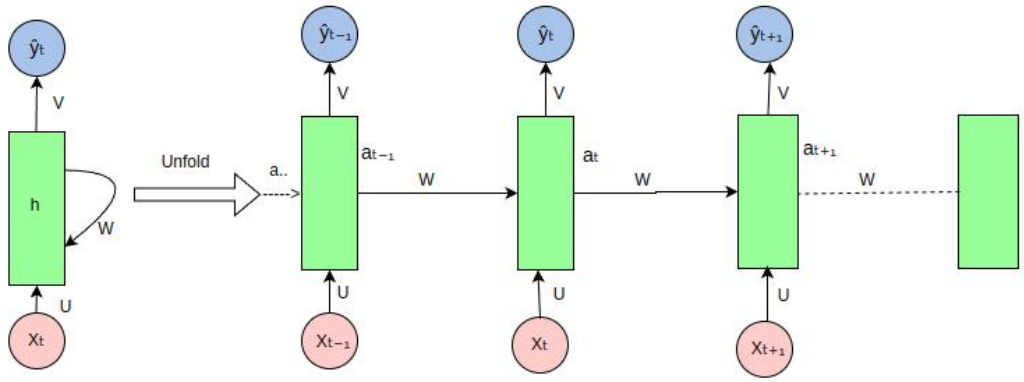


Рисунок 2.2 – Архітектура RNN

- Циклічний зв'язок: Дозволяє RNN зберігати інформацію про попередні стани

- Приховані стани: Зберігають контекстну інформацію про попередні кроки в послідовності

- Переваги:
  - Простота: RNN мають просту архітектуру і легкі для реалізації.
  - Ефективність для коротких послідовностей: RNN добре працюють на коротких послідовностях, де довгострокова залежність не є критичною.
- Недоліки:
  - Обмежена здатність збереження довгострокової залежності: RNN неефективні для довгих послідовностей через свою архітектуру.

2.2.3 GAN

GAN складаються з двох нейронних мереж - генератора та дискримінатора, які змагаються одна з одною. Генератор намагається створювати реалістичні дані на основі випадкового шуму, тоді як дискримінатор намагається відрізнити справжні дані від згенерованих .

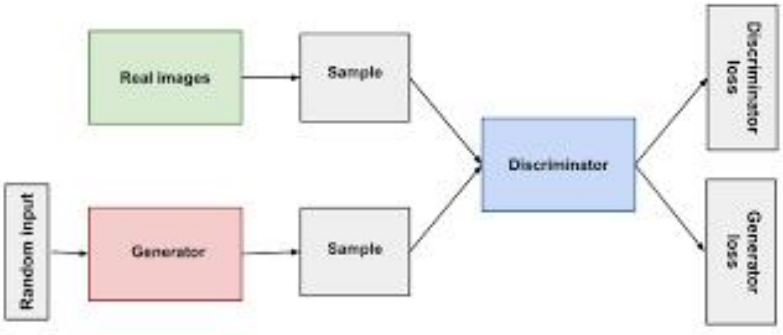


Рисунок 2.3 – Архітектура GAN

- Переваги:
  - Висока якість генерації: GAN можуть створювати дуже реалістичні зразки, що важко відрізнити від справжніх даних .
  - Гнучкість: GAN можна використовувати для генерації різноманітних типів даних, включаючи текст, зображення та аудіо

- Недоліки:
  - Труднощі в тренуванні: Тренування GAN є складним процесом, що вимагає балансування між генератором і дискримінатором і часто генератор може пристосуватися до помилок дискримінатора.
  - Відсутність контролю: Складно контролювати процес генерації, особливо для тексту, де важлива послідовність та логіка.

2.3 Переваги трансформерів над іншими архітектурами

Трансформери, зокрема архітектура, використовувана в моделях GPT (Generative Pre-trained Transformer), значно покращили продуктивність, якість генерації тексту та здатність обробляти довгі залежності у тексті порівняно з традиційними методами, такими як LSTM, RNN та GAN. А особливо комунікувати з новими послідовностями, які модель до цього ще не бачила.

У цьому підрозділі ми детально розглянемо переваги трансформерів, пояснюючи, чому вони стали найбільш поширеними у сфері обробки природної мови (NLP).

2.3.1 Продуктивність

- Паралельна обробка:

  Однією з ключових переваг трансформерів є їх здатність до паралельної обробки даних. На відміну від (RNN) та LSTM, які обробляють дані послідовно, трансформери можуть обробляти всі елементи послідовності одночасно завдяки механізму уваги (attention mechanism). Це значно прискорює процес тренування та генерації тексту.

- RNN і LSTM: Обробляють кожне слово послідовно, що збільшує час обробки для довгих текстів.

- Трансформери: Використовують механізм уваги для паралельної обробки, що зменшує час тренування і дозволяє ефективніше використовувати апаратні ресурси

- Масштабованість:

  Трансформери можуть бути легко масштабовані для роботи з великими обсягами даних. Це дозволяє тренувати моделі на великих корпусах тексту, що підвищує якість генерації та адаптивність моделей до різноманітних задач.

- RNN і LSTM: Складні у масштабуванні через високі вимоги до пам'яті та обчислень, і також тренуються на фіксованій послідовності даних, через що і вихідний текст не може перевищувати задекларовану кількість токенів.

- Трансформери: Легко масштабуються, що дозволяє тренувати моделі на сотнях гігабайтів текстових даних та видавати тексти зовсім різної довжини.

- Якість генерації тексту

Трансформери ефективно зберігають контекстну інформацію, використовуючи механізм уваги, що дозволяє моделі звертати увагу на різні частини вхідного тексту незалежно від його довжини.

Це значно покращує якість генерації тексту, роблячи його більш логічним та узгодженим.

- RNN і LSTM: Мають обмежену здатність зберігати контекстну інформацію на довгих послідовностях через проблеми зникання градієнта.

- Трансформери: Використовують механізм уваги для збереження та обробки контекстної інформації, що забезпечує кращу якість генерації тексту.

- Гнучкість

Трансформери можуть бути адаптовані для виконання різних задач обробки тексту, таких як переклад, реферування, відповідь на запитання та генерація тексту, що робить їх універсальними інструментами для NLP.

- RNN і LSTM: Можуть бути адаптовані для різних задач, але з меншими результатами у порівнянні з трансформерами в додаток їм необхідно набагато більше даних для цього.

- Трансформери: Можуть виконувати широкий спектр задач з високою якістю завдяки своєму гнучкому механізму уваги.

- Навчання та генерація тексту

Навчання трансформерів базується на великому обсязі попередньо навчальних даних (pre-training), а потім налаштування на специфічні задачі (fine-tuning). Це дозволяє моделям отримати загальні знання з великих корпусів тексту і потім адаптуватися до конкретних завдань.

- RNN і LSTM: Потребують великих обсягів даних для кожної конкретної задачі, що ускладнює їх адаптацію до нових задач.

- Трансформери: Використовують попереднє навчання на великих обсягах даних, що дозволяє швидко адаптуватися до нових задач за допомогою невеликої кількості специфічних даних.

- Специфіка тестових даних

Трансформери демонструють високу продуктивність на тестових даних завдяки своїй здатності зберігати контекстну інформацію та обробляти довгострокові залежності. Це дозволяє їм генерувати більш точні та релевантні відповіді.

-RNN і LSTM: Можуть мати проблеми з генерацією точних відповідей на довгі запити через обмежену здатність зберігати контекстну інформацію.

-Трансформери: Демонструють високу точність на тестових даних завдяки своїй архітектурі та ефективному використанню контексту

2.4 Огляд технологій та методів дотренування моделей

У сучасному світі машинного навчання існує безліч інструментів та фреймворків, які допомагають у створенні, тренуванні та дотренуванні великих мовних моделей (LLM). У цьому підрозділі ми детально розглянемо популярні фреймворки на мові Python, такі як TensorFlow, JAX, PyTorch, torchtune, transformers та peft.

Кожен з них має свої унікальні особливості та призначення, що дозволяє ефективно використовувати їх у різних сценаріях.

2.4.1 Tewnsorflow

TensorFlow - це відкритий фреймворк для машинного навчання, розроблений компанією Google. Він був випущений у 2015 році та став одним з найпопулярніших інструментів для створення та тренування моделей глибокого навчання. TensorFlow призначений для побудови та тренування складних нейронних мереж. Він забезпечує гнучкість та масштабованість, що дозволяє використовувати його для різних задач, від розпізнавання образів до обробки природної мови.

З основних можливостей це підтримка як високорівневих, так і низькорівневих API для створення моделей, потужні інструменти для візуалізації та моніторингу тренувального процесу (TensorBoard) та підтримка роботи на різних платформах, включаючи мобільні пристрої та розподілені системи. [8]

2.4.2 Jax

JAX - це фреймворк від Google, створений для ефективних обчислень та автоматичного диференціювання. Він забезпечує високу продуктивність та гнучкість для створення моделей машинного навчання. JAX орієнтований на дослідників та інженерів, які потребують високої продуктивності та гнучкості у своїх моделях. Він дозволяє легко реалізовувати складні обчислювальні графи та виконувати автоматичне диференціювання.

З основних можливостей цього фреймворку можна відмітити швидке автоматичне диференціювання з використанням JIT-компіляції, легка інтеграція з іншими фреймворками та бібліотеками Python, та Підтримка розподілених обчислень та GPU/TPU для прискорення тренувального процесу та легка компіляція коду між TPU та CPU.

2.4.3 PyTorch

PyTorch - це відкритий фреймворк для глибокого навчання, розроблений компанією Facebook (тепер Meta). Він отримав популярність завдяки своїй гнучкості та зручності використання. PyTorch широко використовується для досліджень та розробки нових моделей глибокого навчання. Його динамічний обчислювальний граф дозволяє легко налагоджувати та змінювати моделі під час тренування. З основих можливостей можна назвати динамічний обчислювальний граф, що забезпечує гнучкість та простоту налагодження, це також динамічний обчислювальний граф, що забезпечує гнучкість та простоту налагодження та активну спільноту розробників, які постійно покращують фреймворк[9]

2.4.4 Torchtune

Torchtune це бібліотека з відкритим кодом, створена для полегшення дотренування та налаштування великих мовних моделей (LLM) на основі

PyTorch. Ця бібліотека являє собою автоматизоване налаштування гіперпараметрів: torchtune спрощує процес пошуку оптимальних параметрів для конкретних задач, автоматизуючи процес підбору гіперпараметрів LLM. Дозволяє Додаткове дотренування LLM: torchtune дозволяє дотренувати LLM на власних даних, покращуючи їхню продуктивність у спеціалізованих задачах. torchtune надає зручний інтерфейс для порівняння різних LLM, методів дотренування та конфігурацій.

2.4.5 Transformers

transformers - це бібліотека від Hugging Face, яка надає інструменти для роботи з передтренованими моделями трансформерів. Вона підтримує різні моделі, такі як BERT, GPT, T5 та інші. transformers використовується для дотренування та налаштування передтренованих моделей на специфічних задачах.

Це дозволяє швидко та ефективно адаптувати моделі до нових даних без необхідності тренування з нуля. З основних можливостей можна виділити підтримку великої кількості передтренованих моделей з архітектурою трансформер, простий API для цього дотренування та налаштування моделей на специфічних задачах. Також інтеграцію з іншими фреймворками машинного навчання, такими як PyTorch, Tensorflow. [10]

2.4.6 PEFT

peft - це методика та бібліотека для дотренування моделей з мінімальними змінами параметрів. Вона дозволяє економити обчислювальні ресурси та зменшувати час тренування, налаштовуючи лише невелику частину параметрів моделі. peft використовується для ефективного дотренування

моделей на нових даних, що дозволяє зменшити витрати на обчислення та покращити продуктивність моделей. Основні можливості данного фреймворку це ефективне дотренування моделей з мінімальними змінами параметрів, підтримка різних стратегій дотренування для оптимізації продуктивності та легка інтеграція з іншими фреймворками та бібліотеками для машинного навчання. [11]

2.4.7 Вибір фреймворку

Для даної дипломної роботи ми обрали бібліотеку peft, оскільки вона забезпечує оптимальне співвідношення між ефективністю та продуктивністю для дотренування моделей на специфічних користувацьких даних. Використання peft дозволяє знизити витрати на обчислення та час тренування, зберігаючи високу якість результатів, порівняно із повноцінним тренуванням моделі. Це робить її ідеальним вибором для нашої задачі дотренування великих мовних моделей (LLM) на спеціалізованих даних.

2.5 Методи оцінки роботи моделей

Оцінка продуктивності великих мовних моделей (LLM) є критично важливим етапом у розробці та дотренуванні. В цьому підрозділі ми розглянемо найпопулярніші бенчмарки та метрики, які використовуються для оцінки якості моделей. Серед них: MMLU, Big-bench, GLUE та SuperGLUE, HELM, ROUGE, BLEU Score. Кожен з них має свої специфічні особливості, які дозволяють оцінювати моделі для різних задач. [12]

2.5.1 MMLU

MMLU є бенчмарком, який вимірює здатність моделей виконувати багато різних завдань. Він містить більше 50 завдань, що охоплюють різні сфери знань, такі як математика, природничі науки, історія, мистецтво та інші.

MMLU використовується для оцінки загальної здатності моделі виконувати різноманітні завдання, що вимагають різних навичок і знань. [12]

Основні можливості:

- Оцінка багатозадачності моделей
- Вимірювання продуктивності в різних доменах знань
- Забезпечення комплексного тестування для виявлення сильних та слабких сторін моделей

2.5.2 Big-bench

Big-bench є масштабним бенчмарком, який містить більше 200 завдань, спрямованих на тестування можливостей моделей в різних аспектах, включаючи мовне розуміння, генерацію тексту, відповіді на питання та інші. Big-bench використовується для оцінки широкого спектру можливостей мовних моделей, забезпечуючи глибоке розуміння їх продуктивності у різних контекстах. [13]

Основні можливості:

- Тестування моделей на великій кількості різноманітних завдань
- Виявлення сильних та слабких сторін моделей у різних доменах
- Забезпечення порівняння моделей за багатьма метриками

2.5.3 GLUE

GLUE є популярним бенчмарком для оцінки моделей природної мови. Він складається з декількох завдань, таких як розпізнавання сентименту, парафрази, розпізнавання текстових зв'язків та інші. GLUE використовується для оцінки загальної здатності моделей до розуміння природної мови.

Основні можливості:

- Комплексне тестування моделей на різних задачах природної мови
- Оцінка здатності моделей до розуміння та інтерпретації тексту.

Забезпечення порівняння продуктивності різних моделей

2.5.4 SuperGLUE

SuperGLUE є розширеним варіантом GLUE, який містить більш складні завдання для тестування моделей природної мови. Він включає нові завдання, які вимагають глибшого розуміння тексту та більш складних навичок. SuperGLUE використовується для оцінки просунутих моделей природної мови, які мають вищу здатність до розуміння складних текстових завдань

Основні можливості:

1) Тестування моделей на складніших задачах природної мови
2) Оцінка здатності моделей до глибокого розуміння та інтерпретації тексту.
3) Забезпечення більш високих вимог для сучасних моделей

2.5.5 HELM

HELM є комплексним бенчмарком, який оцінює моделі з різних аспектів, включаючи точність, продуктивність, етичність та інші метрики. Він забезпечує всебічну оцінку моделей, враховуючи різні аспекти їх роботи. HELM використовується для комплексної оцінки моделей, включаючи технічні та етичні аспекти їх продуктивності.

Основні можливості:

- Всебічна оцінка моделей за різними метриками
- Вимірювання технічних та етичних аспектів продуктивності моделей.
- Забезпечення комплексного підходу до оцінки мовних моделей

2.5.6 ROUGE

ROUGE є метрикою, яка використовується для оцінки якості текстових згенерованих відповідей, особливо для завдань реферування. Вона вимірює

кількість співпадінь між згенерованим текстом та референтним текстом. ROUGE використовується для оцінки якості рефератів та згенерованих текстів, особливо в завданнях сумаризації.

Основні можливості:

-Оцінка якості текстових рефератів

-Вимірювання співпадінь між згенерованим та референтним текстом.

-Забезпечення кількісної оцінки якості згенерованого тексту

2.5.7 BLUE

BLEU є метрикою, яка використовується для оцінки якості автоматичного перекладу тексту. Вона вимірює кількість співпадінь між згенерованим перекладом та референтним перекладом. BLEU використовується для оцінки якості автоматичних перекладів тексту.[14]

Основні можливості:

- Оцінка якості автоматичних перекладів

- Вимірювання співпадінь між згенерованим та референтним перекладом.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було детально розглянуто архітектуру трансформерів, яка є основою сучасних великих мовних моделей. Описано ключові компоненти цієї архітектури, такі як механізм уваги, енкодер та декодер, а також їхню взаємодію під час обробки текстових даних. Було проведено порівняння трансформерів з традиційними методами генерації тексту, такими як LSTM, RNN та GAN, виділивши переваги трансформерів у продуктивності, якості генерації тексту, здатності обробляти довгі залежності та навчанні на великих обсягах даних.

Також було розглянуто популярні фреймворки та бібліотеки для роботи з великими мовними моделями, такі як TensorFlow, JAX, PyTorch, torchtune, transformers та peft. Для кожного з них було описано основні можливості, сфери застосування та переваги використання. Зокрема, пояснено вибір бібліотеки peft для даної дипломної роботи, оскільки вона забезпечує оптимальне співвідношення між ефективністю та продуктивністю при дотренуванні моделей на специфічних користувацьких даних.

Наприкінці розділу було охарактеризовано різні бенчмарки та метрики для оцінки якості роботи великих мовних моделей, включаючи MMLU, Big-bench, GLUE, SuperGLUE, HELM, ROUGE та BLEU Score. Кожен з них призначений для оцінки різних аспектів продуктивності моделей, таких як багатозадачність, розуміння природної мови та генерація тексту.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Вибір технік та методів

3.1.1 Вибір моделі для дотренування

Моделі існують у різних розмірах, і чим більша модель, тим кращі результати вона може демонструвати. Проте зі збільшенням розміру моделі збільшується складність обчислень. Наприклад, для тренування моделі з 1 мільярдом параметрів необхідно близько 28 ГБ оперативної пам'яті. Це створює певні обмеження, особливо якщо обчислювальні ресурси обмежені.

Через обмеження власного пристрою в цьому проєкті ми будемо використовувати модель T5 у якої близько 250 мільйонів параметрів. Ця модель є меншою, що полегшує її дотренування та знижує вимоги до обчислювальних ресурсів пристрою. Незважаючи на це, завдяки розумному підходу до написання коду, модель можна легко замінити на більшу або іншу версію в майбутньому, якщо це буде необхідно.

Але так як подібних моделей існують багато, існує покращена версія модель T5, яка набагато краще виконує інструкції – FLAN-T5.

- FLAN-T5: FLAN-T5 (Fine-tuned Language-agnostic Anonymization Transformer) - це спеціалізована версія моделі T5, яка була дотренована з акцентом на певні задачі та контексти. Вона оптимізована для завдань, де важлива адаптація до специфічних даних і контекстів, що робить її більш ефективною в конкретних прикладних сценаріях. FLAN-T5 інтегрує методики, спрямовані на покращення продуктивності в умовах обмежених ресурсів
- Звичайна T5: Модель T5 (Text-To-Text Transfer Transformer) розроблена для перетворення всіх задач обробки природної мови у формат тексту до тексту. Вона відома своєю гнучкістю і здатністю

виконувати широкий спектр задач, включаючи переклад, резюмування, питання-відповідь та інші. Звичайна T5 є більш загальною моделлю і не має спеціалізації на конкретних задачах або контекстах.

Таким чином, вибір FLAN-T5 як базової моделі для дотренування є виправданим, зважаючи на обмеження обчислювальних ресурсів та необхідність адаптації моделі до специфічних користувацьких даних. Цей вибір дозволяє забезпечити баланс між ефективністю дотренування та якістю отриманих результатів.

3.1.2 Вибір підходу для тренування

Враховуючи недостатню обчислювану можливість власного пристрою, ми не можемо використовувати повне дотренування чи тим більше – повне тренування моделі. Саме тому було обрано часткове дотренування, а саме LoRA.

У розділі два я описував багато різних методів дотренування і найєффективнішим з усіх є LoRA – незважаючи на те що ми дотреновуємо лише частину нових параметрів, у порівнянні з повноцінним дотренуванням таких підхід не пасе задніх та показує дуже схожі показники за вищезазначеними метриками якості.[15]

3.2 Специфіка задачі

Задача генерації тексту достатньо багатогранна та обширна, через що на відомих платформах існує багато датасетів, направлених на ту ж генерацію коду, де всього є 2 стовпці : запит користувача та відповідь кодом. Але специфіка нашої задачі це дещо ускладненна версія генерації тексту по попередньому контексту.

Почнемо з того, як повинен виглядати наш запит до майбутньої моделі :

Приклад такого промпту - «Згенеруй текст, направлений на {тема, задана користувачем} спираючись на наступний частотний розподіл характеристик, які повинні бути присутні в цьому тексті:

Мітка №1 : 0.3;

Мітка №2 : 0.2;

Мітка №3 : 0.1;

Мітка №4 : 0.4;»

Як я згадував раніше, подібна задача з частотним розподілом є унікальною, через що схожих навіть сирих даних на конкретну тематику немає. Для базового дотренування великомовної моделі необхідно від 1000 до 1500 семплів даних. Нажаль важко знайти датасет направлений на дещо одну тематику, через що стандартні датасети з платформ HuggingFace чи Kaggle не підходять. Саме тому ми будемо збирати наші самостійно.

3.3 Аналіз та видобуток даних

3.3.1 Аналіз даних для дотренування

Для дотренування великих мовних моделей (LLM) на користувацьких даних важливо вибрати відповідні джерела даних, які будуть релевантними для цільової задачі. У цьому проєкті мій вибір пав на пости на соціо-політичну тематику від популярних інфлюєнсерів у соціальній мережі Twitter (тепер відомій як X). Соціо-політичні пости від інфлюєнсерів є багатим джерелом інформації завдяки їхній актуальності, великій аудиторії та впливу на суспільну думку. Вони часто містять цікаві думки, гострі коментарі та аналітичні спостереження, що робить їх ідеальними для тренування моделей, спрямованих на генерацію текстів з подібними характеристиками.

Крім Twitter, також можна розглянути інші джерела, такі як коментарі під схожими відео на YouTube чи пости у Facebook. Ці платформи надають

доступ до великої кількості користувацького контенту, який може бути корисним для дотренування. Наприклад, коментарі під відео на YouTube часто містять обговорення та зворотний зв'язок щодо політичних тем, а пости у Facebook можуть включати довші дискусії та особисті історії користувачів. Проте, для цього проєкту ми зосередимося на Twitter через його специфічний стиль комунікації та високу динаміку обговорень, що відповідає нашим потребам у тренувальних даних.

3.3.2 Видобуток даних

Концепція датамайнінгу передбачає процес збору, обробки та аналізу великої кількості даних з різних джерел для подальшого використання у навчанні моделей. Видобуток даних з соціальних мереж є важливим етапом для нашого проєкту, оскільки саме ці дані будуть основою для дотренування нашої моделі. Раніше можна було використовувати API Twitter, яке дозволяло збирати до одного мільйона постів за рік. Це забезпечувало достатній обсяг даних для тренування моделей. Однак, після минулорічної купівлі Twitter відомим мільярдером, правила користування API змінилися, що значно ускладнило процес збору даних. Тепер розробники не мають можливості скрапити пости за допомогою офіційного API.

Через ці обмеження нам довелося шукати альтернативні методи збору даних. Відкриті бібліотеки та інструменти для скрапінгу даних з Twitter стали корисними для нашої задачі. З їх допомогою нам вдалося зібрати близько 1500 нових семплів даних. Ці семпли включають пости та коментарі, що відповідають заданій соціо-політичній тематиці, і є достатнім обсягом для початкового етапу дотренування нашої моделі.

3.3.3 Очистка даних

Дані, зібрані з соціальних мереж, часто містять різні специфічні елементи, такі як спеціальні символи, посилання та інші атрибути, які можуть заважати процесу тренування моделі. Тому необхідно провести очистку даних, щоб підготувати їх для дотренування. Одним з основних етапів є видалення спеціальних символів та посилань, які не несуть смислового навантаження і можуть спотворювати результати моделі. Наприклад, хештеги, згадки користувачів та URL-адреси не мають значення для нашої задачі генерації тексту і повинні бути вилучені.

Класична очистка даних у завданнях обробки природної мови (NLP) зазвичай включає видалення стоп-слів, лематизацію та стемінг. Однак, для нашої задачі ці кроки можуть бути менш актуальними. Завдяки архітектурним особливостям трансформерів, модель здатна розпізнавати акценти в тексті і використовувати інформацію, що міститься у стоп-словах та закінченнях слів. Це дозволяє моделі краще розуміти контекст і генерувати більш природний та зв'язний текст.

Таким чином, очистка даних для нашого проєкту включає видалення тільки тих елементів, які явно не сприяють покращенню якості тренування, зберігаючи при цьому всю важливу лінгвістичну інформацію, що може бути корисною для моделі. Цей підхід дозволяє підготувати дані для дотренування з максимальною ефективністю, забезпечуючи високу точність і релевантність згенерованого тексту.

3.4 Класифікація та робота з датасетом

3.4.1 Класифікація датасету

У нашому проєкті необхідно забезпечити класифікацію текстів на основі специфічних характеристик та міток, які будуть визначені користувачами. Використання класичного бінарного або мультикласового класифікатора для цієї задачі не є оптимальним рішенням. Класичні класифікатори, такі як логістична регресія чи нейронні мережі, вимагають великої кількості даних для кожної категорії та наявності чітко визначених міток для тренування. Вони видають конкретні вірогідності для кожної з можливих категорій, але їхня ефективність залежить від попереднього знання всіх можливих міток.

У нашому випадку, ми прагнемо створити універсальне рішення, яке буде здатне класифікувати будь-які тексти за будь-якими мітками, що можуть з'явитися від користувачів. Традиційний мультикласовий класифікатор не здатний задовольнити ці вимоги, оскільки йому знадобиться величезна кількість даних та міток, а навіть це не забезпечить повне покриття всіх можливих категорій. Це обмежує гнучкість та масштабованість системи, що є критичним для нашої задачі.

Для вирішення цієї проблеми ми можемо скористатися іншою великомовною моделлю (LLM), яка спеціалізується на класифікації текстів. Використовуючи бібліотеку transformers, ми маємо можливість вибирати з великої кількості моделей, що вже натреновані на різних задачах та можуть бути адаптовані до нашої специфіки. Зокрема, ми можемо застосовувати моделі, які здатні розуміти контекст тексту і класифікувати його на основі заданих параметрів.

Цей підхід дозволяє нам уникнути необхідності тренувати нову модель з нуля для кожного нового набору міток, що робить систему більш гнучкою та масштабованою. Ми можемо скористатися моделями, такими як BERT чи RoBERTa, які вже довели свою ефективність у задачах класифікації тексту. Ці моделі можуть бути дотреновані або безпосередньо застосовані для класифікації текстів на основі нових, користувацьких міток.

Однак, цей підхід має свої недоліки. Використання готових великомовних моделей для класифікації не завжди забезпечує максимальну точність для специфічних задач. Також, для деяких дуже специфічних категорій може знадобитися додаткове дотренування моделі на нових даних, що може бути ресурсозатратним. Це є слабкою частиною нашого підходу, і тут існує значний потенціал для покращення. Подальше донавчання великомовних моделей на класифікацію різноманітних міток у тексті може значно підвищити точність та універсальність нашого рішення.

Таким чином, обраний нами підхід дозволяє створити гнучку систему, здатну класифікувати тексти на основі будь-яких міток, визначених користувачами. Використання великомовних моделей замість класичних класифікаторів значно розширює можливості нашого проєкту та забезпечує високу точність і релевантність результатів.

3.4.2 Робота з датасетом

Перед тим як далі працювати з текстом, який буде використовуватись для дотренування моделі, необхідно створити готовий текстовий промпт. У нашому випадку він формується за допомогою початкового значення та міток

даних. Це дозволяє нам створити фінальний датасет, який буде використовуватись для навчання моделі.

Машинне навчання будь-якого роду вимагає спеціалізованого підходу до підготовки датасету, зокрема його розбиття на три категорії: тренувальний, валідаційний та тестувальний набори. У нашому проєкті ми розбиваємо дані у пропорції 80-10-10, що дозволяє забезпечити коректну оцінку продуктивності моделі. Тренувальний набір даних використовується для навчання моделі, валідаційний – для підбору гіперпараметрів та перевірки на наявність перенавчання, а тестувальний – для остаточної оцінки якості моделі.

Однак, великомовні моделі, як і будь-які нейронні мережі, не розуміють людської мови в її звичайному вигляді. Вони здатні обробляти лише числові дані. Тому необхідно перетворити весь текст у числове представлення, використовуючи методи ембедингів. Ембединг (embedding) – це спосіб представлення текстових даних у вигляді векторів чисел, які можуть бути оброблені моделями машинного навчання.

Процес ембедингу включає кілька ключових етапів:

- Токенізація – розбиття тексту на окремі частини (токени), які можуть бути словами, підсловами або символами. У нашому випадку ми використовуємо токенізатори, що інтегровані у великомовні моделі, такі як BERT чи T5.
- Перетворення токенів у вектори – кожен токен перетворюється у вектор чисел за допомогою попередньо навчених моделей ембедингу. Ці вектори відображають семантичні значення токенів у багатовимірному просторі.
- Формування вхідних даних – згенеровані вектори використовуються як вхідні дані для моделі. Вони

подаються на вхід нейронної мережі для подальшої обробки та навчання.

Після токенізації та перетворення у вектори, наш датасет складається з числових представлень текстових даних, що готові до використання у процесі навчання моделі.

Розбиття датасету на тренувальний, валідаційний та тестувальний набори забезпечує можливість ретельної перевірки якості моделі на різних етапах її навчання. Пропорція 80-10-10 дозволяє оптимально використовувати дані для навчання, перевірки та оцінки. Тренувальний набір (80%) використовується для безпосереднього навчання моделі, валідаційний набір (10%) допомагає налаштувати гіперпараметри та уникнути перенавчання, а тестувальний набір (10%) забезпечує об'єктивну оцінку кінцевої продуктивності моделі.

Таким чином, процес підготовки датасету включає створення текстових промптів, перетворення тексту у числові представлення за допомогою ембедингів, а також розбиття даних на тренувальний, валідаційний та тестувальний набори. Це забезпечує ефективне навчання моделі, дозволяє уникнути перенавчання та гарантує високу точність і релевантність результатів.

3.5 Тренування моделі

Процес тренування моделі є одним із ключових етапів у створенні ефективної та точної великомовної моделі (LLM). Для дотренування нашої моделі ми використовуємо метод Low-Rank Adaptation (LoRA), що дозволяє значно зменшити обчислювальні витрати та потреби в пам'яті, зберігаючи високу ефективність.

3.5.1 Побудова конфігурації

Метод LoRA дозволяє адаптувати модель до нових задач, змінюючи лише невелику кількість параметрів. У нашому коді ми бачимо використання конфігурації LoRA, яка включає кілька важливих параметрів:

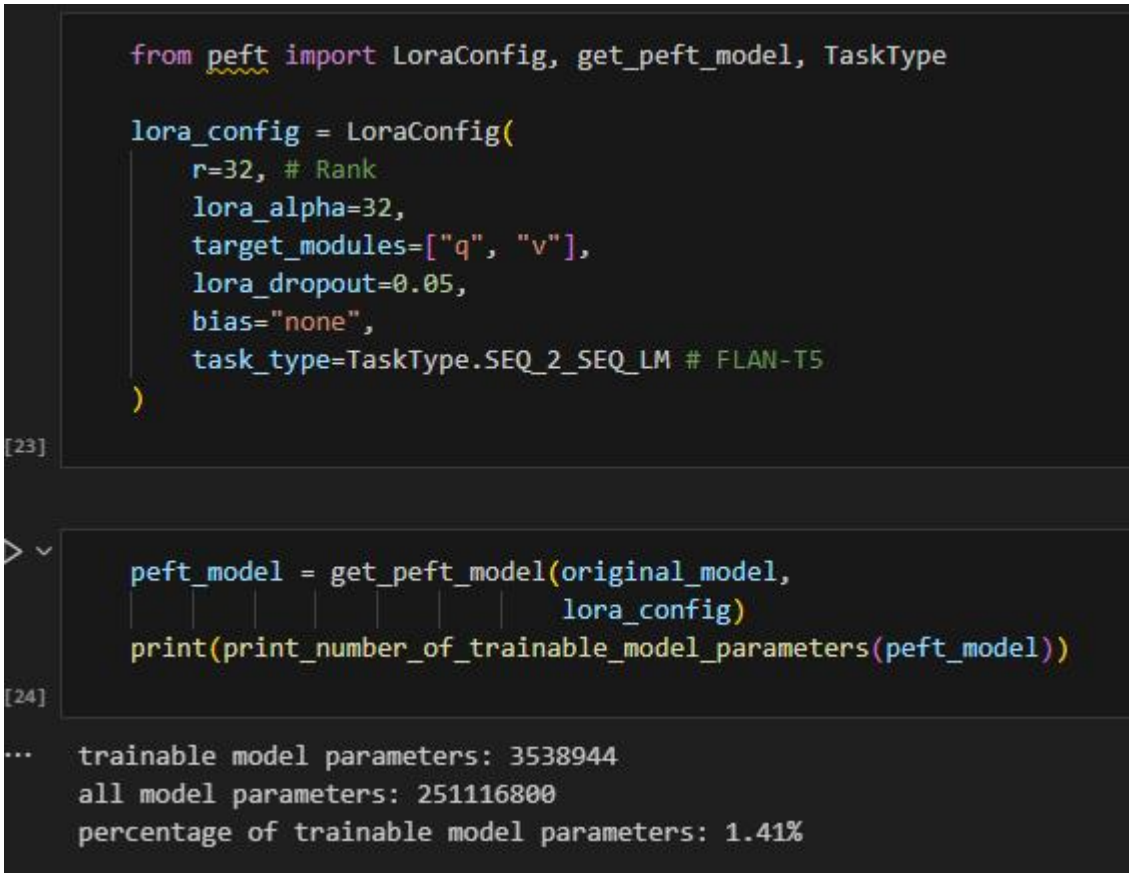


Рисунок 3.1 – Конфігурація дотренувальних параметрів та їх відсоткова кількість

r (Rank): Визначає розмір низькорівневої адаптації. Чим вище значення rank, тим більше параметрів буде змінено, що може покращити якість адаптації, але збільшити обчислювальні витрати.

lora_alpha: Це коефіцієнт масштабування, який регулює величину внеску низькорівневих параметрів у загальну модель. Високі значення дозволяють моделі швидше адаптуватися до нових даних.

target_modules: Визначає конкретні модулі моделі, до яких буде застосована адаптація. У нашому випадку це модулі "q" і "v".

lora_dropout: Встановлює ймовірність dropout для уникнення перенавчання. Значення 0.05 означає, що 5% з'єднань будуть випадково відключені під час тренування.

bias: Визначає, чи буде адаптація включати зміщення (bias). У нашому випадку ми не включаємо зміщення.

task_type: Вказує тип завдання. Для моделі FLAN-T5 ми використовуємо тип завдання SEQ_2_SEQ_LM (послідовність до послідовності).

3.5.2 Логіка дотренування

У процесі тренування ми використовуємо бібліотеку transformers від Hugging Face, яка дозволяє легко адаптувати та дотренувати моделі за допомогою методів на зразок LoRA.

```
peft_training_args = Seq2SeqTrainingArguments(
    output_dir=output_dir,
    auto_find_batch_size=True,
    learning_rate=1e-3,  # Higher learning rate than full fine-tuning.
    num_train_epochs=1,
    logging_steps=10,
    save_steps=500,
    evaluation_strategy="epoch",
    predict_with_generate=True
)

peft_trainer = Seq2SeqTrainer(
    model=peft_model,
    args=peft_training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["valid"]
)

peft_trainer.train()

peft_model_path = "./peft-text-generation-checkpoint-local"
peft_trainer.model.save_pretrained(peft_model_path)
self.tokenizer.save_pretrained(peft_model_path)
```

Рисунок 3.2 – Приклад тренувальних параметрів для моделі

1. learning_rate: Це параметр швидкості навчання, який визначає, наскільки сильно модель буде коригувати свої параметри у відповідь на помилку на кожному кроці. Високе значення може прискорити навчання, але може призвести до нестабільності, тоді як низьке значення забезпечує більш стабільне, але повільніше навчання. У нашому випадку, для LoRA, ми використовуємо відносно високе значення 1e-3, щоб модель швидко адаптувалась до нових даних.
2. num_train_epochs: Кількість епох навчання. Одна епоха відповідає одному повному проходу через весь тренувальний набір даних. Більша кількість епох дозволяє моделі більше навчатися на тренувальних даних, покращуючи її продуктивність, але також збільшуючи ризик перенавчання. У нашому випадку ми використовуємо 1 епоху для LoRA, оскільки метод зосереджується на швидкій адаптації.
3. logging_steps: Визначає частоту логування прогресу навчання. Значення 10 означає, що кожні 10 кроків буде збережено інформацію про прогрес.
4. ave_steps: Визначає, як часто зберігати модель під час навчання. Значення 500 означає, що модель буде збережена кожні 500 кроків.
5. evaluation_strategy: Встановлює стратегію оцінювання моделі. Значення epoch означає, що оцінювання буде виконуватись після кожної епохи.
6. predict_with_generate: Визначає, чи буде використовуватись метод генерації для оцінки продуктивності моделі. Це важливо для моделей послідовності до послідовності, як FLAN-T5.

Процес тренування включає кілька кроків. Ми повинні перш за все підготувати наші дані, тому що без даних наша модель не зможе нічого навчитися. Ми використовуємо тренувальну частину датасету, після кожної епохи перевіряючи результат на валідаційному тренуванні . Далі до нашої тренувальної моделі застосовується конфігурація LoRa, яка додає якусь частину параметрів до початкової моделі.

І потім, нарешті використовуємо «Seq2SeqTrainer», забезпечуючи ефективне тренування та оцінювання моделі.

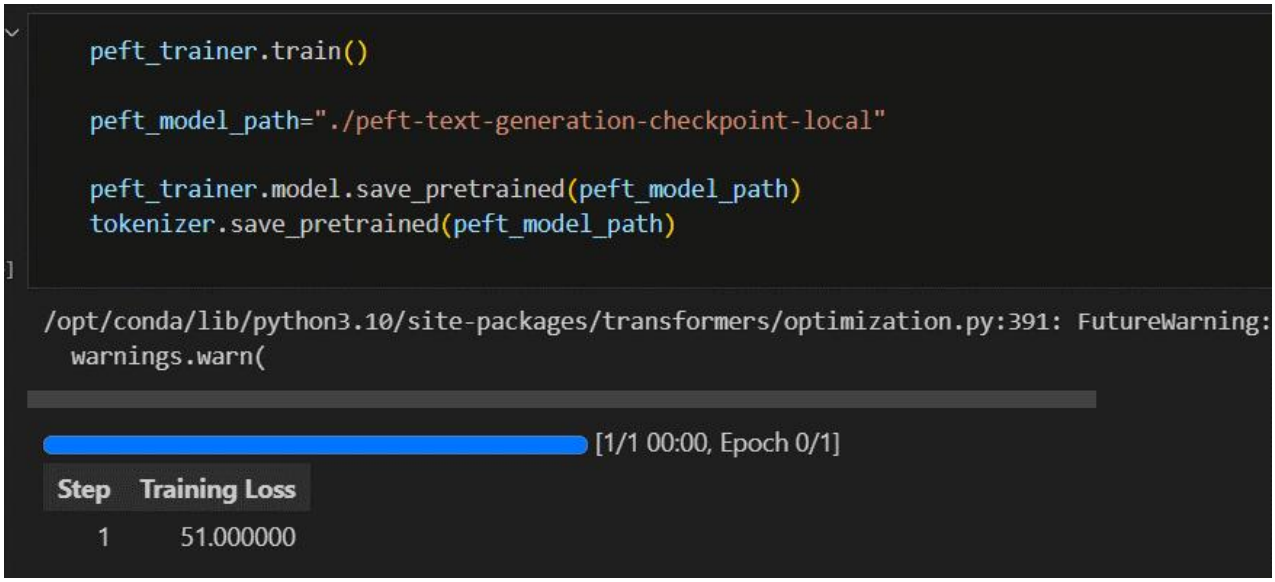


Рисунок 3.3 – Тренування моделі

Після закінчення тренування, модель та її токенайзер, тобто модель яка перетворює текст на вектор ембедингів для розуміння тексту самою моделлю, зберігаються для подальшої роботи або оцінювання.

ВИСНОВОК ДО РОЗДІЛУ 3

В цьому розділі ми зосередилися на ключових аспектах створення та навчання нашої великомовної моделі для генерації тексту згідно з заданим частотним розподілом характеристик.

Спочатку було зроблено обґрунтований вибір моделі для дотренування. Враховуючи обмеження обчислювальних ресурсів, було обрано модель T5, але згодом перевага була надана її покращеному варіанту - FLAN-T5. Ця модель є більш ефективною та спеціалізованою для адаптації до специфічних даних і контекстів, що робить її оптимальним вибором для нашої задачі.

Далі, враховуючи недостатню обчислювальну потужність, ми обрали часткове дотренування за допомогою методу LoRA. Цей підхід дозволяє досягти схожих результатів з повноцінним дотренуванням, але при менших витратах ресурсів.

Наступним кроком було визначення специфіки нашої задачі та збір відповідних даних для дотренування. Оскільки подібна задача з частотним розподілом є унікальною, ми не знайшли готових датасетів і вирішили збирати їх самостійно. Для цього було обрано пости на соціо-політичну тематику від популярних інфлюєнсерів у соціальній мережі Twitter. Ці дані були відповідними для нашої задачі та надали достатній обсяг для початкового етапу дотренування.

Після збору даних, ми провели їх очистку та підготовку. Було видалено спеціальні символи, посилання та інші елементи, що не несуть смислового навантаження. Також, враховуючи архітектурні особливості трансформерів, ми вирішили не видаляти стоп-слова та проводити лематизацію, оскільки модель здатна розпізнавати акценти в тексті і використовувати інформацію, що міститься у стоп-словах та закінченнях слів.

РОЗДІЛ 4. ПЕРЕВІРКА ТА АНАЛІЗ РОБОТИ

4.1 Реальні тести та порівняння з іншими моделями

Перед тим як далі працювати з текстом, який буде використовуватись для дотренування моделі, необхідно створити готовий текстовий промпт. У нашому випадку він формується за допомогою початкового значення та міток даних.

В якості поверхової оцінки роботи ми візьмемо схожі моделі з Hugging Face та надамо всім моделям однаковий промпт :

«INPUT PROMPT:

Create a tweet that reflects the proportional representation of the following features. Each feature's presence in the tweet should correspond to its percentage in the overall probability distribution (where the sum of all feature probabilities equals 1). Please ensure that the content and tone of the tweet are influenced accordingly by these proportions.

Feature Probability Distribution:

{'bullying': 0.014132256619632244, 'insult': 0.032489363104104996, 'profanity': 0.03951292484998703, 'sarcasm': 0.07885083556175232, 'threat': 0.07880470156669617, 'exclusion': 0.09896454215049744, 'porn': 0.01809712126851082, 'spam': 0.026300324127078056, 'lgbt': 0.04045671224594116, 'black lives matter': 0.02411658875644207, 'ecological activism': 0.018695704638957977, 'feminism': 0.022569293156266212, 'politics': 0.0156567804515361 8, 'positive': 0.4322446882724762, 'negative': 0.0591081939637661}

Tweet:»

4.1.1 Gemma-1.1-2b-it

Результати моделі: «Representation of Features in Communication:

While navigating diverse experiences, we acknowledge the presence of various elements that shape communication.

Bullying, insults, and threats carry significant weight, impacting 21.6% of interactions.

While positivity shines brightly at 43%, negative sentiments occupy 5.9% of communication.

Join the dialogue and explore the other features that influence our conversations! #ProportionalRepresentation #CommunicationDynamics<eos> »

Ця модель являє собою в 10 разів більшою за нашу, яка має 250 мільйонів параметрів, але справляється досить непогано.

4.1.2 Phi-3

Результати роботи моделі : “Positive vibes only! 🌟✨ Our community thrives on inclusivity and respect. 🤝💖 Let's keep the conversation constructive and uplifting. 🗣️📢 #Positive #CommunityFirst #RespectAll “

Ця модель має ще більше параметрів а справляється набагато краще. Але її текст не такий великий.

4.1.3 Qwen1.5-4b-chat

Результат роботи моделі :

«"From bullying to positive, our probability distribution captures the diverse range of topics that impact our online experiences. Let's come together to create a safe and inclusive online community. #PositiveBalance # RespectEachOther"»\

Як ми бачимо, в цьому випадку результат не дуже підходящий і не відповідає правильному результату.

4.1.4 Оригінальна модель

Так як ми дотренували модель, необхідно порівняти її х власним попередником. Результати роботи моделі :

«@samantha_samantha i'm not sure if i'm a sarcasm or a feminism or a feminism»

4.1.5 Дотренована модель:

Наша модель навчалась 1 єпосі, але це не завадило їй досягти результатів, схожих на вище вказані. Результати дотренованої моделі:

«I'm thrilled to finally share my new show with you all! 🌟 After two years of hard work, it's ready. Episodes will drop every Thursday! Don't miss out! #Exciting #PositiveVibes»

4.2 Оцінка роботи моделі

У розділі 2 я описували багато різних метрик які можна застосовувати і в моєму випадку ми будемо використовувати метрику ROUGE, а саме її підвиди – Rouge1, Rouge2, RougeL. Ці підвиди метрики оцінюють текст з різної послідовності. Rouge1 та Rouge2 оцінюють точність уні та біграм відповідно, RougeL – оцінює точність найдовшої спільної послідовності, враховуючи послідовність слів.

```python
rouge = evaluate.load('rouge')

original_model_results = rouge.compute(
    predictions=original_model_generation,
    references=human_baseline_generation[0:len(original_model_generation)],
    use_aggregator=True,
    use_stemmer=True,
)

instruct_model_results = rouge.compute(
    predictions=instruct_model_generation,
    references=human_baseline_generation[0:len(instruct_model_generation)],
    use_aggregator=True,
    use_stemmer=True,
)

peft_model_results = rouge.compute(
    predictions=peft_model_generation,
    references=human_baseline_generation[0:len(peft_model_generation)],
    use_aggregator=True,
    use_stemmer=True,
)

print('ORIGINAL MODEL:')
print(original_model_results)
print('INSTRUCT MODEL:')
print(instruct_model_results)
print('PEFT MODEL:')
print(peft_model_results)
```

```
ORIGINAL MODEL:
{'rouge1': 0.187769756385947, 'rouge2': 0.04849999999999999, 'rougeL': 0.1403101433337705}
INSTRUCT MODEL:
{'rouge1': 0.41026607717457186, 'rouge2': 0.17840645241958838, 'rougeL': 0.2977022096267017}
PEFT MODEL:
{'rouge1': 0.3725351062275605, 'rouge2': 0.12138811933618107, 'rougeL': 0.27620639623170606}
```

Рисунок 4.1 – Оцінка моделі за допомогою метрик.

Як ми можемо побачити на зображені, наша модель(PEFT MODEL) у порівнянні із повністю дотренованої версії моделі (INSTRUCT MODEL) має непогані результати, це та сама перевага методу PEFT над повним дотренуванням – задіюємо меньше ресурсів але отримуємо негірші результати

Також потрібно оцінити абсолютне покращення дотренованої моделі над звичайною

```
print("Absolute percentage improvement of PEFT MODEL over ORIGINAL MODEL")

improvement = (np.array(list(peft_model_results.values())) - np.array(list(original_model_results.values())))
for key, value in zip(peft_model_results.keys(), improvement):
    print(f'{key}: {value*100:.2f}%')
```

```
Absolute percentage improvement of PEFT MODEL over ORIGINAL MODEL
rouge1: 17.47%
rouge2: 8.73%
rougeL: 12.36%
```

Рисунок 4.2 – Покращення дотренованої моделі над звичайною.

І також потрібно зрозуміти скільки відсотків точності ми програємо у порівнянні з повним дотренуванням

```
print("Absolute percentage improvement of PEFT MODEL over INSTRUCT MODEL")

improvement = (np.array(list(peft_model_results.values())) - np.array(list(instruct_model_results.values())))
for key, value in zip(peft_model_results.keys(), improvement):
    print(f'{key}: {value*100:.2f}%')
```

```
Absolute percentage improvement of PEFT MODEL over INSTRUCT MODEL
rouge1: -1.35%
rouge2: -1.70%
rougeL: -1.34%
```

Рисунок 4.3 – Покращення дотренованої моделі над повністю дотренованої.

ВИСНОВОК ДО РОЗДІЛУ 4

Для поверхової оцінки роботи ми взяли схожі моделі з Hugging Face та надали всім моделям однаковий промпт. Було протестовано моделі Gemma-1.1-2b-it, Phi-3, Qwen1.5-4b-chat, оригінальну модель та дотреновану модель.

Результати показали, що зі збільшенням параметрів моделі, вони можуть справлятися і без дотренування. Модель Gemma-1.1-2b-it, яка в 10 разів більша за нашу, справляється досить непогано. Модель Phi-3, яка має ще більше параметрів, справляється набагато краще, але її текст не такий великий. Модель Qwen1.5-4b-chat не показала таких високих результатів.

Оригінальна модель та дотренована модель також були протестовані. Дотренована модель, яка навчалась лише 1 єпохою, показала результати, схожі на результати більших моделей.

Для більш детальної оцінки роботи моделі, ми використали метрику ROUGE та її підвиди - Rouge1, Rouge2, RougeL. Ці метрики дозволили оцінити точність уніграм, біграм та найдовшої спільної послідовності, враховуючи послідовність слів.

Результати показали, що наша модель у порівнянні з повністю дотренованою версією моделі має непогані результати. Також ми оцінили абсолютне покращення дотренованої моделі над звичайною та зрозуміли, скільки відсотків точності ми програємо у порівнянні з повним дотренуванням.

Можна зробити висновок, що дотренована модель може бути ефективним рішенням для генерації тексту згідно з заданим частотним розподілом характеристик, при цьому вимагаючи менше ресурсів, ніж повністю дотренована модель.

ВИСНОВКИ

У цій дипломній роботі ми дослідили великі мовні моделі (LLM) та їхню адаптацію до спеціалізованих задач бізнесу. Початковий огляд LLM, їхньої історії, призначення та типових задач, пояснив необхідність дотренування цих моделей для адаптації до специфічних галузей та завдань.

Ми детально розглянули архітектуру трансформерів, яка є основою сучасних LLM, та провели порівняння з традиційними методами генерації тексту. Використання популярних фреймворків та бібліотек, таких як TensorFlow, JAX, PyTorch, torchtune, transformers та peft, дозволило нам оптимізувати процес дотренування моделей на специфічних користувацьких даних.

У практичній частині роботи, ми зосередилися на створенні та навчанні нашої великомовної моделі для генерації тексту згідно з заданим частотним розподілом характеристик. Вибір моделі FLAN-T5 та часткове дотренування за допомогою методу LoRA, дозволили нам досягти схожих результатів з повноцінним дотренуванням, але при менших витратах ресурсів.

Проведення поверхової та детальної оцінки роботи моделі, показало, що дотренована модель може бути ефективним рішенням для генерації тексту згідно з заданим частотним розподілом характеристик, при цьому вимагаючи менше ресурсів, ніж повністю дотренована модель.

Загалом, ця дипломна робота закладає міцний фундамент для розуміння великих мовних моделей, їхніх можливостей, обмежень та необхідності адаптації шляхом дотренування до конкретних вимог. Результати нашої роботи можуть бути корисними для бізнесу та інших організацій, які прагнуть використовувати LLM для вирішення спеціалізованих задач.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Wolf T., Tunstall L., Werra L. v. Natural Language Processing with Transformers, Revised Edition. O'Reilly Media, Incorporated, 2022.
2. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., Polosukhin I. Attention Is All You Need / arXiv.org. — 2017. — URL: https://arxiv.org/abs/1706.03762 (Дата : 06.05.2024).
3. Minaee S., Mikolov T., Nikzad N., Chenaghlu M., Socher R., Amatriain X., Gao J. Large Language Models: A Survey / arXiv.org. — 2024. — URL: https://arxiv.org/abs/2402.06196 (Дата звернення: 13.05.2024).
4. Fine-tune a pretrained model. Hugging Face – The AI community building the future. URL: https://huggingface.co/docs/transformers/training (date of access: 01.06.2024).
5. Naveed H., Khan A. U., Qiu S., Saqib M., Anwar S., Usman M., Akhtar N., Barnes N., Mian A. A Comprehensive Overview of Large Language Models / arXiv.org. — 2023. — URL: https://arxiv.org/abs/2307.06435 (Дата звернення: 15.05.2024).
6. Foster D. Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play. O'Reilly Media, Incorporated, 2023.
7. Sri M. Practical Natural Language Processing with Python. Berkeley, CA : Apress, 2021. URL: https://doi.org/10.1007/978-1-4842-6246-7 (дата звернення: 12.05.2024).
8. API Documentation | TensorFlow v2.16.1. TensorFlow. URL: https://www.tensorflow.org/api_docs (дата звернення: 03.06.2024).
9. PyTorch documentation – PyTorch 2.3 documentation. PyTorch. URL: https://pytorch.org/docs/stable/index.html (дата звернення: 02.06.2024).
10. Transformers. Hugging Face – The AI community building the future. URL:

https://huggingface.co/docs/transformers/index (дата звернення: 20.05.2024).

11. Xu L., Xie H., Qin S.-Z. J., Tao X., Wang F. L. Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment / arXiv.org. — 2023. — URL: https://arxiv.org/abs/2312.12148 (Дата звернення: 25.05.2024).
12. Guo Z., Jin R., Liu C., Huang Y., Shi D., Supryadi, Yu L., Liu Y., Li J., Xiong B., Xiong D. Evaluating Large Language Models: A Comprehensive Survey / arXiv.org. — 2023. — URL: https://arxiv.org/abs/2310.19736 (Дата звернення: 19.05.2024).
13. Hu T., Zhou X.-H. Unveiling LLM Evaluation Focused on Metrics: Challenges and Solutions / arXiv.org e-Print archive. — 2024. — URL: https://arxiv.org/html/2404.09135v1 (Дата звернення: 26.06.2024).
14. Wang Y., Jiang J., Zhang M., Li C., Liang Y., Mei Q., Bendersky M. Automated Evaluation of Personalized Text Generation using Large Language Models / arXiv.org. — 2023. — URL: https://arxiv.org/abs/2310.11593 (Дата звернення: 24.05.2024).
15. Hu E. J., Shen Y., Wallis P., Allen-Zhu Z., Li Y., Wang S., Wang L., Chen W. LoRA: Low-Rank Adaptation of Large Language Models / arXiv.org. — 2021. — URL: https://arxiv.org/abs/2106.09685 (Дата звернення: 31.05.2024).

ДОДАТОК 1

Система переднавчання великомовних моделей на основі даних користувача

СХЕМА РОБОТИ АЛГОРИТМУ ПІДГОТОВКИ ДАНИХ

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2024 рік

					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Басенко Н.Р			**Система переднавчання великомовних моделей на основі даних користувача** Схема роботи алгоритму підготовки даних	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Русінов В.В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-05	
Н. Контр.		Ковальчук О.М					
Затвердив							

ДОДАТОК 2

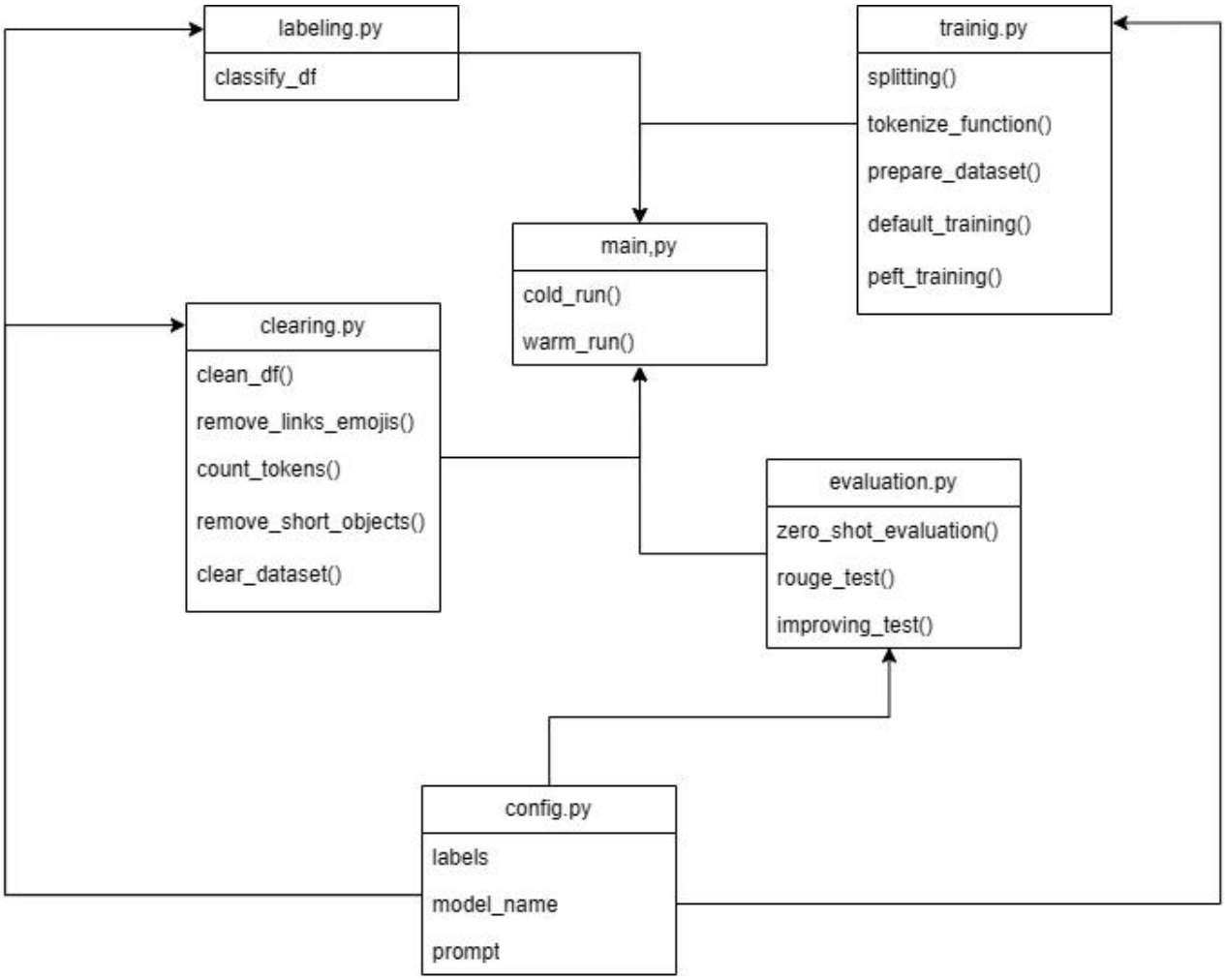
Система переднавчання великомовних моделей на основі даних користувача

ФУНКЦІОНАЛЬНА СХЕМА
(ДІАГРАМА КЛАСІВ)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2024 рік



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Басенко Н.Р.			**Система переднавчання великомовних моделей на основі даних користувача** Функціональна схема (діаграма класів)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Русінов В.В					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-05	
Н. Контр.		Ковальчук О.М					
Затвердив							

ДОДАТОК 3

Система переднавчання великомовних моделей на основі даних користувача

АЛГОРИТМ ДІЙ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ – 2024 рік

					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Басенко Н.Р.			**Система переднавчання великомовних моделей на основі даних користувача** Алгоритм дій програмного забезпечення	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Русінов В.В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, І0-05	
Н. Контр.		Ковальчук О.М					
Затвердив							

ДОДАТОК 4

Система переднавчання великомовних моделей на основі даних користувача

ТЕКСТ ПРОГРАМНОГО КОДУ

ІАЛЦ.467200.007 Д4

Аркушів 11

Київ – 2024 рік

```
import pandas as pd
import numpy as np
from datasets import load_dataset
from datasets import DatasetDict
import os

from transformers import AutoModelForSeq2SeqLM, AutoTokenizer,
GenerationConfig, TrainingArguments, Trainer
from transformers import Seq2SeqTrainer, Seq2SeqTrainingArguments
from peft import LoraConfig, get_peft_model, TaskType

import torch
import time
import evaluate

from training_script.config import START_PROMPT, END_PROMPT, MODEL,
COLUMN_NAME, labels

class Trainer():
    def __init__(self):
        self.dataset_path = os.path.join(os.getcwd(), "data",
"labeled_data.csv")
        self.split_ratios = {
                                'train': 0.8,
                                'valid': 0.1,
                                'test': 0.1
                            }
        self.ds_splits = None

        self.start_prompt = START_PROMPT
        self.end_prompt = END_PROMPT
        self.text_column = COLUMN_NAME
        self.tokenizer = AutoTokenizer.from_pretrained(MODEL) # take
out like parametr to Trainer(tokenizer)
        self.labels = labels

        self.output_dir = f'./task-generation-training-
{str(int(time.time()))}'


    def info(self):
        print("WORK DIR for TRAINING: ", os.getcwd())
```

```
        print("Dataset path: ", self.dataset_path)

    def splitting(self, split_ratios: dict = None) -> DatasetDict:
        """

        """
        if split_ratios is None:
            split_ratios = self.split_ratios


        ds = load_dataset('csv', data_files=self.dataset_path)
        assert sum(split_ratios.values()) == 1, "Split summ should be
equal to 1"

        train_test_split = ds['train'].train_test_split(test_size=(1
- split_ratios['train']), seed=1706)
        test_valid_split =
train_test_split['test'].train_test_split(test_size=(split_ratios['te
st'] / (split_ratios['test'] + split_ratios['valid'])), seed=1706)

        self.ds_splits = DatasetDict({
            'train': train_test_split['train'],
            'valid': test_valid_split['train'],
            'test': test_valid_split['test']
        })

        return self.ds_splits

    def tokenize_function(self, example):
        start_prompt = self.start_prompt
        end_prompt = self.end_prompt

        labels = example.copy()
        tweet = labels.pop(self.text_column)

        prompt = start_prompt + str(labels) + end_prompt
        tokenized_input = self.tokenizer(prompt, padding="max_length",
truncation=True)
        tokenized_labels = self.tokenizer(tweet, padding="max_length",
truncation=True)

        tokenized_input["labels"] = tokenized_labels["input_ids"]
```

```
        return tokenized_input


    def prepare_dataset(self):

        ds_splits = self.splitting()
        tokenized_datasets = ds_splits.map(self.tokenize_function)
        tokenized_datasets =
tokenized_datasets.remove_columns(self.labels)

        print(f"Shapes of the datasets:")
        print(f"Training: {tokenized_datasets['train'].shape}")
        print(f"Validation: {tokenized_datasets['valid'].shape}")
        print(f"Test: {tokenized_datasets['test'].shape}")

        return tokenized_datasets

    def print_number_of_trainable_model_parameters(self, model):
        trainable_model_params = 0
        all_model_params = 0
        for _, param in model.named_parameters():
            all_model_params += param.numel()
            if param.requires_grad:
                trainable_model_params += param.numel()
        return f"trainable model parameters:
{trainable_model_params}\nall model parameters:
{all_model_params}\npercentage of trainable model parameters: {100 *
trainable_model_params / all_model_params:.2f}%"

    def default_training(self, output_dir: str = None):
        if output_dir is None:
            output_dir = self.output_dir

        tokenized_datasets = self.prepare_dataset()
        original_model = AutoModelForSeq2SeqLM.from_pretrained(MODEL,
torch_dtype=torch.bfloat16)

        training_args = Seq2SeqTrainingArguments(
        output_dir=output_dir,
        learning_rate=1e-5,
        num_train_epochs=5,
        weight_decay=0.01,
```

```
        logging_steps=10,  # Adjust logging steps to be reasonable
        per_device_train_batch_size=8,  # Adjust batch size as needed
        per_device_eval_batch_size=8,   # Adjust batch size as needed
        save_steps=500,                 # Save checkpoint every 500
steps
        evaluation_strategy="epoch",    # Evaluate at the end of each
epoch
        predict_with_generate=True      # Use generate to calculate
metrics
        )

        trainer = Seq2SeqTrainer(
            model=original_model,
            args=training_args,
            train_dataset=tokenized_datasets['train'],
            eval_dataset=tokenized_datasets['valid']
        )

        trainer.train()

    # def evaluate(self):
    #     # Load the model from the training directory
    #     instruct_model =
AutoModelForSeq2SeqLM.from_pretrained(self.output_dir,
torch_dtype=torch.bfloat16)


    def peft_training(self, output_dir: str = None):
        if output_dir is None:
            output_dir = self.output_dir

        tokenized_datasets = self.prepare_dataset()
        original_model = AutoModelForSeq2SeqLM.from_pretrained(MODEL,
torch_dtype=torch.bfloat16)

        lora_config = LoraConfig(
                                r=64, # Rank
                                lora_alpha=32,
                                target_modules=["q", "v"],
                                lora_dropout=0.05,
                                bias="none",
```

```
                                task_type=TaskType.SEQ_2_SEQ_LM #
FLAN-T5
                            )

        peft_model = get_peft_model(original_model, lora_config)

print(self.print_number_of_trainable_model_parameters(peft_model))

        peft_training_args = Seq2SeqTrainingArguments(
            output_dir=output_dir,
            auto_find_batch_size=True,
            learning_rate=1e-3,  # Higher learning rate than full
fine-tuning.
            num_train_epochs=1,
            logging_steps=10,
            save_steps=500,
            evaluation_strategy="epoch",
            predict_with_generate=True
        )

        peft_trainer = Seq2SeqTrainer(
            model=peft_model,
            args=peft_training_args,
            train_dataset=tokenized_datasets["train"],
            eval_dataset=tokenized_datasets["valid"]
        )

        peft_trainer.train()

        peft_model_path = "./peft-text-generation-checkpoint-local"
        peft_trainer.model.save_pretrained(peft_model_path)
        self.tokenizer.save_pretrained(peft_model_path)

import pandas as pd
from transformers import pipeline
from tqdm import tqdm

from training_script.config import labels, COLUMN_NAME

class Labeler():
    def __init__(self):
        self.classifier = pipeline("zero-shot-classification")
```

```
        self.labels = labels
        self.text_column = COLUMN_NAME

    def classify_df(self, tweets_df: pd.DataFrame, labels: list[str]
= None, text_column: str=None) -> pd.DataFrame:
        """
        Classifying tweets in a dataframe by your labels.

        :param tweets_df: DataFrame containing the tweets.
        :param text_column: Name of the column containing the tweet
texts.
        :param labels: List of labels to classify the tweets.
        :return: DataFrame with original tweets and their
classification scores.
        """

        if labels is None:
            labels = self.labels
        if text_column is None:
            text_column = self.text_column

        results = []

        for index, row in tqdm(tweets_df.iterrows(),
total=tweets_df.shape[0], desc="Classifying tweets"):
            tweet = row[text_column]
            result = self.classifier(tweet, labels)

            result_dict = {label: score for label, score in
zip(result['labels'], result['scores'])}
            # print("Tweet - ", tweet)
            # print(result_dict)
            result_dict[text_column] = tweet

            results.append(result_dict)

        results_df = pd.DataFrame(results)

        cols = [text_column] + [col for col in results_df if col !=
text_column]
        results_df = results_df[cols]
```

```
        return results_df

import pandas as pd
import re
from pathlib import Path
from transformers import AutoTokenizer

from training_script.config import MODEL, COLUMN_NAME


#create class for it

class Cleaner() :
    def __init__(self):
        self.model_name = MODEL
        self.tokenizer =
AutoTokenizer.from_pretrained(self.model_name) # take out like
parametr to Cleaner(tokenizer)
        self.min_tokens = 10
        self.column_name = COLUMN_NAME

    def clean_df(self, df: pd.DataFrame, column_name: str) ->
pd.DataFrame:
        return df.dropna(subset=[column_name])

    def info(self):
        print("MODEL NAME : ", self.model_name)

    def remove_links_emojis(self, text: str) -> str:
        """
        Deleted links and emojis from a given text.

        Args:
            text: Текст, из которого нужно удалить ссылки и смайлики.

        Returns:
            The text with links and emojis removed.
        """

        # Remove links that start with http or https
        text = re.sub(r"(https?://\S+)", "", text)
```

```
        # Remove links that start with www
        text = re.sub(r"www\.\S+", "", text)

        # Remove other urls without http(s) or www but with known
TLDs
        text =
re.sub(r"\b\S+\.(com|org|net|io|ai|co|us|uk|ru|de|fr|jp|cn|in|br|pl|e
s|it|nl|se|no|fi|dk|ch|be|at|au|ar|mx|ca|ua|kz|by|ie|nz|sg|hk|edu|gov
|mil|eu|info|biz|me|tv|pro|name|club|xyz|website|online|site|tech|spa
ce|store|fun|press|digital|rocks|life|today|guru|ninja|agency|news|ne
twork|media|social|family|blog|vip|web|group|company|link|work|help|p
ics|photo|photos|world|place|host|wiki|click|city|center|chat|church|
zone|land|guide|farm|education|events|direct|community|catering|busin
ess|builders|boutique|bargains|academy)\b", "", text)

        # Remove emojis
        text = re.sub(r'[^\w\s\.]+', '', text)

        return text

    def count_tokens(self, text: str) -> int:
        """
        Counts the number of tokens in a given text.

        Args:
            text: The text to count tokens for.

        Returns:
            The number of tokens in the text.
        """

        input_ids = self.tokenizer.encode(text, return_tensors="pt")
        num_tokens = input_ids.shape[1]
        return num_tokens

    def remove_short_objects(self, df: pd.DataFrame, column: str,
min_tokens: int = 10) -> pd.DataFrame:
        """
        Deletes objects from a column if their length is less than a
specified number of tokens.

        Args:
```

```
            df: Pandas DataFrame.
            column: The name of the column from which short objects
should be deleted.
            min_tokens: The minimum number of tokens that an object
should have.

        Returns:
            DataFrame with short objects removed.
        """
        df_filtered = df[df[column].apply(lambda text:
self.count_tokens(text) >= min_tokens)]
        return df_filtered


    def clear_dataset(self, dataset: pd.DataFrame, column_name: str =
None) -> pd.DataFrame :

        if column_name is None:
            column_name = self.column_name

        if dataset[column_name].isna().sum() > 0:
            dataset = self.clean_df(dataset, column_name)
        dataset[column_name] =
dataset[column_name].apply(self.remove_links_emojis)
        df_cleared = self.remove_short_objects(dataset.copy(),
column_name, self.min_tokens)
        dataset_path = "../data/cleared_dataset_today.csv"
        df_cleared.to_csv(dataset_path, index=False)
        return df_cleared

import json
from ntscraper import Nitter
import pandas as pd
import time
from pathlib import Path

def scrape_twitts(username: str,
                  count: int = 150,
                  threshold: float = 0.5,
                  continue_scrape: bool = True) -> list[dict]:
    """
    collecting tweets from username
```

```
    """
    tweets = Nitter().get_tweets(username, mode="user", number=count)
    all_tweets = tweets['tweets']
    if continue_scrape:
        while len(all_tweets) < threshold*count:
                # time.sleep(60)
                tweets = Nitter().get_tweets(username, mode="user",
number=count)
                all_tweets = tweets['tweets']
    return tweets

def saving_data(username: str, tweets: list[dict]):
    """
    Saves tweets to a JSON file in the "user_tweets" folder.

    Args:
        username: The Twitter username.
        tweets: A list of dictionaries representing tweets.
    """

    # Create the "user_tweets" directory if it doesn't exist.
    user_tweets_dir = Path("user_tweets")
    user_tweets_dir.mkdir(parents=True, exist_ok=True)

    # Form the filename.
    filename = f"2_{username}_{len(tweets)}.json"

    # Save the data to a file.
    with open(user_tweets_dir / filename, "w") as file:
        json.dump(tweets, file)

def collect_twitts(users: list[str], save_tweets: bool = True) ->
pd.DataFrame:
    dataset = []
    for username in users:
        print(f"Processing {username}")
        try:
            user_twitter = scrape_twitts(username)
        except Exception as ex:
            print("EXCEPTION : ", ex)
            try:
                user_twitter = scrape_twitts(username)
```

```
            except Exception as ex:
                print("EXCEPTION №2 : ", ex)
                continue

        if save_tweets: # сделать по другому
            saving_data(username, user_twitter['tweets'])

        for tweet in user_twitter["tweets"]:
            # # Skip retweets
            # if tweet["is-retweet"]:
            #   continue

            # Tweet text
            tweet_text = tweet["text"]

            # If the tweet text is short, add the text the user is
replying to
            if count_tokens(tweet_text) < 10 and tweet["quoted-post"]:
                reply_text = tweet["quoted-post"]["text"]
                tweet_text = f"{tweet_text}, about '{reply_text}'"

            dataset.append({
                'username': tweet['user']['username'],
                'profile_id': tweet['user']['profile_id'],
                'tweet': tweet_text
            })
    return pd.DataFrame(dataset)
```