Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Автоматизація процесів підтримки роботи над науковими публікаціями на основі систем ШІ (Automation of support processes for work on scientific publications based on AI systems)

Текстова частина до кваліфікаційної роботи
за спеціальністю «Інженерія Програмного Забезпечення» - 121

Керівник кваліфікаційної роботи
Доцент, к.н.
Афонін А. О.
_____ (Підпис)
“___” _____ 2024 року
Виконав студент
ІПЗ-4 Іванов А. В
“___” _____ 2024 року

Київ 2024

Календарний план виконання роботи

№	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	26.10.2024	
2.	Огляд літератури за темою роботи	10.11.2024	
3.	Проведення дослідження	01.12.2024	
4.	Написання програмного застосунку	01.01.2024	
5.	Написання текстової частини	04.04.2024	
6.	Передзахист кваліфікаційної роботи	14.05.2024	
7.	Захист кваліфікаційної роботи	27.05.2024	

Студент _____

Керівник _____ “_____”_____ 2024

Вступ

Актуальність теми

Сучасний світ науки і техніки розвивається дуже швидко. Щороку кількість наукових публікацій зростає, і це створює нові виклики для вчених. Обробка цієї величезної кількості інформації стає все важчою і займає багато часу. Автоматизація у створенні наукових робіт може значно скоротити цей час і підвищити продуктивність досліджень.

Автоматизація процесів підтримки роботи над науковими публікаціями, яка базується на системах штучного інтелекту, є дійсно актуальною. Це пов'язано з декількома важливими факторами. По-перше, щороку публікується все більше статей, і обробляти їх вручну стає неймовірно складно. По-друге, дослідники потребують швидкого доступу до потрібної інформації, щоб робити обґрунтовані висновки. Крім того, наукові тексти часто містять складну термінологію, і методи природньої обробки мови можуть автоматизувати цей процес. Сучасні досягнення в галузі штучного інтелекту, такі як глибоке навчання і великі мовні моделі, дозволяють розвивати такі системи. Інтелектуальні системи допомагають зменшити ручну працю і зосередитись на більш креативних і аналітичних завданнях.

Тому створення веб-сервісу для автоматизації роботи над науковими публікаціями є дуже актуальним на данний момент. Це відповідає великим сучасним потребам наукової спільноти і допомагає ефективніше використовувати час і ресурси, покращуючи якість досліджень.

1.2 Мета та завдання дослідження

Мета дослідження полягає у створенні веб-сервісу, який за допомогою штучного інтелекту здатний автоматично обробляти наукові тексти. Це має в собі підсумовування тексту, пошук сайтів з корисною інформацією, а також аналіз настрою тексту та інші функції, що сприяють зручній роботі дослідників.

Для досягнення цієї мети були поставлені наступні завдання:

1. Розробка архітектури веб-сервісу:
    - Створення структури, що інтегрує різні модулі обробки тексту.
    - Забезпечення зручного та водночас просто інтерфейсу.
2. Реалізація модуля пошуку за ключовими словами:

- Використання методів природної обробки мови (NLP) для того щоб виділяти ключові слова з тексту.
- Застосування алгоритму TF-IDF, для сортування ключових слів, по тому наскільки вони важливі.
- Інтеграція з пошуковим двигуном Google для знаходження потрібних сайтів.

3. Розробка модуля сумаризації тексту:
    - Навчання та використання T5 для автоматичної сумаризації тексту.
    - Забезпечення високої точності а також релевантності в згенерованих сумаризацій.
4. Розробка модуля аналізу настрою тексту:
    - Навчання та використання моделі BERT для того щоб оцінювати тональності тексту (наприклад: позитивна, негативна, нейтральна).
    - Аналіз точності та ефективності роботи моделі.
5. Навчання та тестування моделей:
    - Збір та підготовка великої бази навчальних даних для кожної з моделей.
    - Проведення навчання моделей та оцінка їх ефективності за допомогою метрик якості.
6. Інтеграція та тестування веб-сервісу:
    - Інтеграція всіх модулів в один єдиний веб-сервіс.
    - Проведення тестування веб-сервісу на реальних даних, для його майбутньої оцінки продуктивності та коректності роботи.

1.3 Методологія дослідження

Методологія дослідження базується на поетапній реалізації та інтеграції різних технологій штучного інтелекту, для створення корисного веб-сервісу з автоматизації обробки наукових текстів. Далі описано основні етапи дослідження та методи, які були використані на кожному з них:

1. Аналіз вимог та постановка задачі:
    - Визначення основних функцій, які має виконувати веб-сервіс, наприклад: сумаризація тексту, пошук релевантної інформації, аналіз настрою тексту.
    - Проведення огляду існуючих рішень та технологій у сфері обробки природної мови (NLP) та штучного інтелекту.
2. Розробка архітектури системи:
    - Створення загальної архітектури веб-сервісу, що включає різні модулі для обробки тексту.
    - Визначення технологій та інструментів, які будуть використані для розробки кожного з модулів (Python, TensorFlow, PyTorch тощо).
3. Реалізація модулів обробки тексту:
    - Модуль пошуку за ключовими словами:
        - Використання алгоритмів NLP для виділення ключових слів з тексту.
        - Застосування методу TF-IDF для сортування ключових слів за їх важливістю.
        - Інтеграція з Google Search API для знаходження потрібних сайтів.
    - Модуль сумаризації тексту:
        - Використання моделі T5 для автоматичної сумаризації тексту.

        - Навчання моделі на великому наборі даних для забезпечення високої точності суммаризацій.
    - Модуль аналізу настрою тексту:
        - Використання моделі BERT для оцінки тональності тексту.
        - Тренування моделі на різноманітних текстових корпусах для точного визначення настрою (позитивний, негативний, нейтральний).
4. Збір та підготовка даних:
    - Збір великого обсягу текстових даних з наукових публікацій для навчання та тестування моделей.
    - Попередня обробка даних, включаючи очищення тексту, токенізацію та нормалізацію.
5. Навчання та тестування моделей:
    - Проведення навчання кожної з моделей на попередньо підготовлених даних.
    - Використання метрик якості (наприклад, точність, повнота, F1-міра) для оцінки на скільки модель ефективна.
    - Тестування моделей на реальних даних для перевірки їх продуктивності а також коректності роботи.
6. Інтеграція модулів у веб-сервіс:
    - Інтеграція всіх модулів у єдину систему, таким чином забезпечення їх взаємодії і узгодженості роботи.
    - Розробка користувацького інтерфейсу для забезпечення зручності використання веб-сервісу.
7. Тестування та валідація веб-сервісу:
    - Проведення тестування веб-сервісу на різних наборах даних для чіткої оцінки його продуктивності.

- Збір зворотного зв’язку від користувачів для виявлення можливих проблем і покращення функціональності системи.

Завдяки такому комплексного підходу та сучасних технологій штучного інтелекту, ця методологія дозволяє створювати дійсно ефективний веб-сервіс для автоматизації обробки наукових текстів, що має значно полегшити роботу людей які хочуть створити наукові публікації.

2. Огляд літератури

2.1 Сучасні підходи до автоматизації процесів обробки текстової інформації

В нашому сучасному саіті обробка текстової інформації і в цілому різної інформації стає все більш важливою, через дуже великий обсяг даних. Автоматизація цих процессів, дозволяє набагато швидше, аналізувати дуже багато тексту. Давайте розглянемо основні підходи які використовуються для автоматизації обробки тексту

По-перше це NLP, цей підхід використовує алгоритми і моделі, для того щоб дуже добре розуміти людську мову. NLP включає в себе токенізацію, роспізнавання сутностей, та зв’язків між текстом, частиномовний аналіз і інше.

Також, метод TF-IDF, це статистичний метод, який потрібен для того щоб зрозуміти мажливість слів у якомусь документі, цей метод допомагає виділяти дуже важливі значумі слова, що дозволяє як можна ефективніше шукати дуже корисну інформацію, а також проводити аналіз текстів.

Окрім цього є моделі на основі глибокого навчання, вони значно збільшили можливості людства працювати з обробкою текстів, для прикладу BERT, GPT-2, і T5, використовуються для дуже великої

кількості різних задач, включаючи генерацію тексту, суммаризацію тексту, аналіз настрою, і багато цншого, ми можемо це побачити на прикладі чатугпт від компанії openAI. Ці моделі вчаться на дуже великому обсязі данних , і за рахунок цього і своїх розмірів показують дуже високу точність в різних завданнях.

Також окрім цього є суммаризація тексту, аналіз настрою тексту, а також іфнормаційний пошук та знаходження сутностей, всі ці процесси допомагають знаходити корисну інформацію в в дуже вкликих обсягах тексту. Всі ці алгоритми часто інтегруються в популярних веб-системах пошуку, такі як Google, Yahoo та інші

2.2 Використання NLP у наукових дослідженнях

Природна обробка мови (NLP) на сьогоднішній момент, дуже важливий інструмент у різних наукових дослідженнях. Її можливості допомагають обробляти дуже великі обсяги тексту, що дійсно спрощує роботу дослідників. Розглянемо основні способи, як в цілому NLP використовується у наукових дослідженнях.

1. **Сумаризація текстів**. Цей інструмент дозволяє створювати маленькі резюме великих текстів. Це особливо корисно коли йде робота з оглядами літератури або великими звітами, де важливо дійсно швидко отримати основну ідею. Завдяки NLP можна автоматично генерувати узагальнення, які передають суть тексту.
2. **Аналіз тональності**. NLP може визначати тон тексту, чи є він позитивним, негативним або нейтральним. Це дуже корисно для розуміння настрою авторів у наукових публікаціях або відгуках наукової спільноти. Такий аналіз може вказати на загальне ставлення до певної теми або дослідження.

Завдяки цим можливостям, NLP робить наукові дослідження більш ефективними та продуктивними. Воно допомагає дослідникам швидко знаходити необхідну інформацію, аналізувати великі обсяги текстів та отримувати корисні висновки. Це значно спрощує роботу та економить час, дозволяючи зосередитися на більш важливих аспектах дослідження.

2.3 Основи методів TF-IDF та їх застосування

TF-IDF (Term Frequency-Inverse Document Frequency) – це простий, але ефективний метод для визначення важливості слів у документі. Цей метод широко використовується в інформаційному пошуку та обробці текстів.

Основи TF-IDF

1. **Term Frequency (TF)**. Цей показник, потрібнен для того щоб показувати, як часто слово зустрічається в документі. Чим частіше слово використовується в тексті, тим вищий TF. Формула виглядає так: $$TF(t,d) = \frac{\text{Кількість появ слова t в документі d}}{\text{Загальна кількість слів в документі d}}$$
2. **Inverse Document Frequency (IDF)**. Цей показник оцінює, наскільки рідко слово зустрічається у всіх документах. Чим рідше слово зустрічається, тим вищий IDF. Формула виглядає так: $$IDF(t) = \log\left(\frac{\text{Загальна кількість документів}}{\text{Кількість документів, що містять слово t}}\right)$$
3. **TF-IDF**. Комбінуючи TF і IDF, отримуємо TF-IDF, який показує важливість слова в конкретному документі в контексті всіх

документів. Формула виглядає так:

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

Застосування TF-IDF

Пошук інформації. TF-IDF аналізує та виділяє найсуттєвіші слова у тексті, що сприяє полегшенню пошуку інформації. Наприклад, в пошукових системах цей підхід використовується для знаходження найбільш важливих документів за запитом користувача.

Групування текстових матеріалів. Метод TF-IDF може бути застосований для класифікації подібних документів разом. Це корисно для структурування обсягових текстових матеріалі, таких як новинні статті або наукові публікації.

Аналіз тексту. TF-IDF допомагає виділити ключові слова, що спрощує отримання загальної картини змісту документа. Це може бути корисно для аналізу великих текстових даних у різних галузях, від маркетингу до наукових досліджень.

Сумаризація тексту. Наш Метод TF-IDF можна використовувати для автоматичного створення підсумку тексту, виділяючи найважливіші речення. Це також дуже корисно для швидкого ознайомлення з вмістом великих документів.

TF-IDF – це в цілому дуже потужний інструмент, який дуже спрощує роботу з текстами, допомагаючи знаходити і аналізувати важливу інформацію. Завдяки своїй простоті та ефективності, цей метод став світовим стандартом у не малої кількості завдань з обробки текстів.

2.4 Огляд великих лінгвістичних моделей (LLM)

Великі лінгвістичні моделі, це такі алгоритми, які зазвичай використовуються для того щоб оброблювати природню мову, останнім часом, вони стали досить популярні, через те що розуміють, та створюють текст на людській, нашій мові. Тут я хочу подати декілька визначених моделей і їх можливе застосування

Перше це GPT 2 і 3 моделі, вони створені OpenAI, які можуть генерувати текст, на основі того який був початковий введений фрагмент, тобто наприклад GPT-3 це понад 170 мільярдів параметрів, що робить його одним з найпутжніших моделей, це дозволяє йому робити дуже багато корисних речей, такі як наприклад: писати статті, відповідати на запитання, перекладати тексти і навіть таку річ, як творити вірші.

Подруге ми можемо застосовувати автоматизацію створення контенту за рахунок нього, також є чат-боти, вони використовуються для того, щоб відповідати на різні запитання клієнтів.

Окрім цього він часто допомогає в освітній программі, за допомогою нього можна зручно навчатись, не дивлячись на проблеми,

Також один із досить важливих LLM це берт, він розроблений гугл, і працює за рахунок двонаправлених трансформаторів, для аналізу контексту в реченні, це значить що він враховує слова які знаходяться біля слова, для того щоб краще зрозуміти суть. Його застосовують для пошукових систем, він покращує точність результату пошуку, також для аналізу настрою, він допомагає визначити чи є текст позитивним чи негативним, чи нейтральним. Окрім цього він може використовуватись в відповідях на запитання, це наприклад також використовується в

багатьох важливих системах,
Також є T5 (Text-to-Text Transfer Transformer) це досить корисна та важлива для різного характеру навчання моедль від гугл, яка робить всі завдання з обробки мови у форматі текст в текст, це означає, що будь які задачі від перекладу і аж до суммаризації, оброблюються як завдання перетворення тексту. ЇЇ використовують для схожих речей як і попередні нейронні мережі, але в неї є своя різниця. Наприклад переклад тестів, суммаризація, виправлення граматики.
Проблеми які виникають з нейромережею:
Зазвичай вона потребує великі обсяги даних, навчання таких моделей потребує дуже великої кількості даних і обчислювальних ресурсів, це робить їх тренування дуже дорогими, а також затрачує дуже багато часу на це навчання. Також є проблема з етичним питанням, існує досить багато побоювань що нейромережі можуть почати створювати дуже багато фейкових новин, або іншу шкідливу інформацію, яка може завдати великих проблем великій кількості людей, також часто в них є пролеми з точністю, оскільки навіть в найкращих моделей, є багато проблем, і особливо важливо слідкувати за помилками в критичних облаястях, таких як наприклад меодицина, біологія, або ж правове поле. В цілому LLM це дуже зручний спосіб обробки інформації, також вони продовжують розвиватись, вони мають дуже великий потенціал, для подальшого вдосконалення, для того щоб покращии різні сфери нашого життя.

3.1 Загальна архітектура системи

Архітектура веб-сервісу для підтримки роботи над науковими публікаціями складається з двох основних частин: перша це фронтенд (web app) на JavaScript, розроблений у WebStorm, та

бекенд на Python з використанням PyTorch. Далі детальніше про ці компоненти та їх взаємодію.

Фронтенд (Web App)

1. **Структура проекту**:
    - **Папка src**: Усі основні файли і компоненти веб-застосунку знаходяться тут.
    - **Файли**:
        - **Analyze.js** і **Analyze.css**: Компонент для аналізу тексту.
        - **App.js** і **App.css**: Головний компонент застосунку.
        - **Home.js**: Домашня сторінка.
        - **Search.js** і **Search.css**: Компонент для пошуку інформації.
    - **Додаткові файли**:
        - **index.js** і **index.css**: Початкові файли для рендерингу застосунку.
        - **logo.svg**: Логотип застосунку.
        - **reportWebVitals.js** і **setupTests.js**: Файли для тестування і моніторингу продуктивності.

Бекенд (Python та PyTorch)

1. **Структура проекту**:
    - **Папка djangoProject**:
        - **asgi.py**, **wsgi.py**, **settings.py**, **urls.py**: Конфігураційні файли Django проекту.
        - **celery.py**: Файл для конфігурації Celery, якщо потрібна асинхронна обробка завдань.
    - **Папка search**: Модуль для пошуку інформації.
        - **models.py**, **views.py**, **urls.py**, **admin.py**: Основні файли для реалізації пошукового функціоналу.
    - **Папка sentiment_project**: Модуль для аналізу настрою тексту.

- **sentiment_analysis.py**: Основний файл для аналізу настрою.
- **models.py**, **views.py**, **urls.py**: Файли для реалізації функціоналу аналізу настрою.

3.1.1 Середовище розробки WebStorm

Для розробки фронтенду нашого веб-сервісу ми використовували WebStorm. Це середовище розробки (IDE) від JetBrains, яке дуже зручне для роботи з JavaScript та багатьма іншими веб-технологіями. WebStorm надає безліч корисних функцій, наприклад таких як: автозаповнення коду, підсвітка синтаксису, інтеграція з системами контролю версій та багато іншого.

Використання WebStorm дозволило нам швидко і ефективно створювати компоненти для нашого застосунку. Наприклад, завдяки автозаповненню коду ми могли уникнути багатьох дрібних помилок, які часто виникають при ручному введенні. Крім того, інтеграція з Git спростила відстеження змін і спільну роботу над проектом.

Ще одна корисна функція WebStorm - це можливість запускати і тестувати застосунки прямо в IDE. Це дозволило нам швидко перевіряти зміни і бачити результат одразу, без необхідності постійно перемикатися між різними вікнами.

Іноді виникали дрібні проблеми, такі як збої або зависання, але загалом робота в WebStorm була дуже продуктивною. Для тих, хто

працює з фронтендом, це дійсно хороший інструмент, який значно полегшує процес розробки.

3.1.2 Середовище розробки PyCharm

Для розробки бекенду мого веб-сервіси я використовува PyCharm, це дуже зручна IDE від JetBrains, вона створена спеціально для пайтона і тому в ній дуже приємно та зручно з ним працювати

По-перше, пайчарм має атовзаповнення коду, це дозволяє дуже швидко писати код і в деяких моментах це дуже добре було видно, особливо у порівнянні з іншими IDE, оскільки іноді автозаповнення може починати працювати прям з нового рядка, де навіть нічого немає, що може бути дуже зручно в цілому.

Також, пайчарм дає багато різних інструментів, для налагодження коду. Окрім цього досить корисна підтримка віртальних середовищ, це дозволяє ізолювати залежності проекту і уникати досить багато конфліктів між різними бібліотеками, наприклад для свого проекту я створив своє середовище, з усим необхідним для спрощення роботи

3.2.1 Алгоритм виділення ключових слів

Виділення ключових слів з тексту — це дуже важливий крок, який допомагає знайти нам основні теми та ідеї в документі. Нижче описано досить простий алгоритм, який використовує методи NLP

та TF-IDF для виділення ключових слів. Для прикладу ось як це працює:

1. Попередня обробка тексту:

- Токенізація: Спочатку ми розбиваємо текст на окремі слова або токени. Для прикладу, текст: "Це приклад тексту для аналізу" перетворюється на ["Це", "приклад", "тексту", "для", "аналізу"].
- Видалення стоп-слів: Стоп-слова — це загальні слова, які не несуть великого значення для розуміння тексту, наприклад такі як "і", "але", "на". Ми видаляємо їх зі списку токенів. Наприклад: ["Це", "для"] видаляються, залишаючи ["приклад", "тексту", "аналізу"].
- «Лематизація або стемінг»: Зводимо слова до їхньої основної форми, щоб уникнути дублювання. Наприклад, "аналізується" стає "аналіз".

2. «Розрахунок TF-IDF»:

- Частота терміну (TF): Розраховуємо, як часто кожне слово з’являється в тексті. Наприклад, якщо слово "текст" з’являється 3 рази у документі з 100 слів, його TF буде 0.03.
- Обернена частота документів (IDF): Розраховуємо, як рідко слово зустрічається в наборі документів. Наприклад, якщо слово "текст" з’являється в 10 з 1000 документів, його IDF буде (log(1000 / 10) = 2).

$$TF\text{-}IDF = TF \times IDF$$

- «Обчислення TF-IDF»: Комбінуємо TF і IDF для кожного слова, щоб отримати його важливість. Слова з високим значенням TF-IDF вважаються ключовими.

3. «Вибір ключових слів»:

- Визначаємо поріг або кількість ключових слів, які потрібно виділити. Наприклад, вибираємо топ-5 слів з найвищим значенням TF-IDF.

- Відбираємо слова з найвищим TF-IDF. Наприклад, якщо у нас є список [("текст", 0.6), ("аналіз", 0.5), ("приклад", 0.4)], ми вибираємо всі три слова як ключові.

Приклад роботи

«Вхідний текст»: "Нейронні мережі використовуються для аналізу великих даних і створення передбачень."

1. «Попередня обробка»:

- Токенізація: ["Нейронні", "мережі", "використовуються", "для", "аналізу", "великих", "даних", "і", "створення", "передбачень"]

- Видалення стоп-слів: ["Нейронні", "мережі", "використовуються", "аналізу", "великих", "даних", "створення", "передбачень"]

- Лематизація: ["Нейронний", "мережа", "використовуватися", "аналіз", "великий", "дані", "створення", "передбачення"]

2. Розрахунок TF-IDF:

- TF для "Нейронний" = 1/10 = 0.1

- IDF для "Нейронний" = (log(1000 / 50) = 1.3)

- TF-IDF для "Нейронний" = 0.1 * 1.3 = 0.13

І так для кожного слова.

3. Вибір ключових слів:
   - Сортуємо слова за значенням TF-IDF.
   - Вибираємо слова з найвищим значенням TF-IDF.

Цей алгоритм дозволяє автоматично виділяти найважливіші слова з тексту, що значно спрощує аналіз великих обсягів інформації і пошук релевантних документів.

3.3 Модуль витягу та сумаризації інформації

3.3.1 Використання моделі T5 для сумаризації тексту

Модуль для створення, та узагальнення інформації допомагає перетворення великих обсягів тексту до маленьких і зрозумілих резюме. Також це дуже корисно, коли потрібно дуже швидко отримати основні ідеї з довгих статей або документів, або іншого тексту. Для цього ми використовуємо модель T5 (Text-to-Text Transfer Transformer) від Google. Ось як це працює.

Що таке T5?

T5 — це модель, яка може перетворювати текст з одного формату у інший. Наприклад, вона може брати дійсно довгий текст та створювати невеличке резюме. Модель була навчена на достотньо великій кількості текстових даних та може виконувати різноманітні завдання, включаючи сумаризацію.

В цілому, як використовувати T5 для сумаризації?

1. Спочатку ми дробимо текст:

• Спочатку необхідно попередньо обробити текст для сумаризації, будь то стаття, звіт або будь-який інший документ.

• Важливо мати чистий текст без серйозних проблем чи надмірних символів.

2. Налаштування моделі: можливо скористатися готовою моделлю T5 з бібліотек Hugging Face для того, щоб швидко почати роботу.

3. Сумаризація тексту: Подайте текст моделі для його узагальнення. І тоді модель поверне невеличкий підсумок, що мість основни ідеї тексту.

Помилки та труднощі

1. Якість узагальнення:

• Часом можуть бутти пропущенì важливì деталì або створено неточне резуме. Тому бажано перевіряти результат в ручному.

2. Обмеження на кількість символів у тексті:

• Модель може працювати лише з обмеженою кількістю символів у тексті. Якщо текст дуже довгий, його треба розбити на частини.

3. Залежність від якості вхідних даних:

• Якщо текст має багато помилок або є незрозумілим, модель може створити неправильне резюме.

Використання моделі T5 для узагальнення тексту робить процес аналізу великих обсягов інформації швидшим та зручним. Це сприяє ефективному використанню часу та зусиль, дозволяючи сфокусуватися на основних ідеях.

3.4 Модуль аналізу тональності тексту

3.4.1 Використання моделі BERT для оцінки тональності

Модуль аналізу емоційного забарвлення тексту допомагає розпізнати емоційний стан тексту - позитивний, негативний або нейтральний. Для

цього ми використовуємо модель BERT, розроблену компанією Google. BERT добре розуміє контекст і може точно визначити емоційне забарвлення.

Принцип роботи BERT полягає в тому, що спочатку ми подаємо текст на обробку моделі, яка аналізує його і дає оцінку емоційного забарвлення. Наприклад, якщо текст передає позитивні почуття, модель визначить його як позитивний. У випадку негативного тексту, наприклад з скаргами, модель також це видентифікує.

Ось простий приклад. Маємо речення “Цей фільм був дуже захоплюючим і радуючим.” Подаємо його на обробку у модель, і BERT повертає результат про позитивне емоційне забарвлення тексту. Інша ситуціая: “Обслуговування багатьма бачиться жахливим та дуже повільним.” Модель визначить це як негативний контекст.

Для тонкощей налаштування BERT користуються бібліотекою Hugging Face. Декілька строк коду легко інтегрують модель у наш застосунок. Головне - правильно попереджений текст без зайвої символіки. Тоді модель працює точно і швидко.

3.4.2 Архітектура та особливості роботи моделі BERT

Bert – є однією з найвідоміших моделей для того, щоб обробляти тексти природньої мови, тобто використовувати NLP. Ця модель була розроблена гуглом, і представлена в 2018 році. Основне, як працює Bert включає наступне:

Архітектура BERT:

По-перше берт базується на трансформерах, яка використовує механізм уваги до себе (self-attention) для того щоб оброблювати текст, трансфорери допомогають враховувати контекст не тільки основний а й той що зліва та зправа, від нашого конкретного слова, що робить їх дуже потужними, для того щоб оброблювати текст, Окрім цього вона двонаправлена, тобто однією з ключових властивостей берт є ця двонаправленість, оскільки вона може оброблювати текст з обох боків кожного слова в реченні, що якраз відрізняється від того як зазвичай роблять однобічні моделі, які дивляться тільки з одного боку слова. Також, під час тренування, деякі слова в реченні, замінюються масками, наприклад [MASK], і тоді завдання моделі передбачити ці маски, це досить добре допомогає моделі зрозуміти суть навколишніх слів і покращує її можливості в генерації якоїсь конкретної суті. Також, важливо пригадати Next Sentence Prediction, NSP, що в перекладі означає вгадування наступного речення, тобто берт дивиться чи слідує одну речення за іншим, це допомогає моделі краще розуміти як будуються зв’язки між реченнями.

Також в особливості роботи берту ми можемо відзначити декілька основних етапів, тобто наприклад Попереднє тренування, де ми тренуємо модель, на великій кількості тексту, наприклад на вікіпедії, за допомогою двох завдань: MLM і NSP, це дозволяє моделі в цілому зрозуміти як працює мова, і отримати загальне розуміння того як будувати речення. Потім Модель налаштовується тонко, для вже досить конкретних задач, наприклад це може бути: класифікація тексту, виявлення сутностей, аналіз чутливості тощо. Це відбувається шляхом тренування моделі на невеликих наборах данних, спеціально підібраних для цих завдань. Також важливо сказати про універсальність моделі, завдяки свої архітектурі та тим, як він тренується, берт показує високу

ефективність на різних задачах по роспізнаванню слів і він може бути легко адаптований для різних завдань з мінімальними змінами в архітектурі.

Переваги берта:

Берт добре розуміє розширене розуміння контексту, за рахунок своєї двонаправленості, та уваги до взаємозв`язків між словами, також він досить гнучкий, ця модель може швидко ажаптуватись до широкого спечктру завдань з роспізнаванням слів, з невеликими зусилляи для тонкого налаштування, при цьому попри просте використання він має досить високу точність, на різних задачах нлп, та частіше краще справляється ніж інші моделі.

4.1 Збір та підготовка даних

Навчання моделей починається зі збору та підготовки даних. Це дуже важливий етап, бо якість даних сильно вплине на подальші результати. Спочатку треба знайти джерела, звідки будемо брати інформацію. Це можуть бути статті, книги, веб-сайти чи інші тексти, які відповідають темі дослідження.

Коли ми зібрали дані, потрібно їх підготувати. Це означає, що треба видалити зайві символи, помилки і привести текст до досить звичайної форми. Наприклад, можна видалити HTML-теги з веб-сторінок, також прибрати дублікаті слів або виправити різні граматичні помилки. Це робиться для того, щоб модель не плуталася і як можна краще розуміла текст.

Далі важливо розділити дані на тренувальні валідаційні і тестові. Тренувальні дані використовуються для навчання моделі, а тестові - для перевірки, наскільки добре вона працює, також валідаційні дані потрібні для того щоб зрозуміти наскільки добре тренується модель. Зазвичай, 80% даних іде на тренування, а 20% - на тестування. Це допомагає оцінити, чи модель не просто запам'ятала дані, а дійсно навчилася їх аналізувати.

Інколи доводиться збирати дуже багато даних, особливо якщо тема складна або нова. Але це варто того, бо чим більше якісних даних, тим краще працює модель. Підготовка даних може займати багато часу, але це необхідний крок для отримання гарних результатів.

4.2 Процес навчання моделей

Навчання моделей - це цікавий і водночас непростий процес. Давайте розглянемо, як ми навчаємо модель BERT для аналізу тональності та модель T5 для роботи з даними з S2ORC.

Спочатку поговоримо про BERT. Щоб навчити BERT розпізнавати сантименти, нам потрібні дані з позначками: позитивний, негативний, нейтральний. Ми беремо тексти з відгуками, коментарями чи ще іншими джерелами, де чітко видно емоційний тон. Ці тексти обробляємо та подаємо в модель. BERT натомість використовує свій механізм уваги, щоб зрозуміти контекст кожного слова і на основі цього визначити тональність всього тексту. Як приклад візьмемо текст "Цей продукт мені дуже

сподобався!" - він буде позначений як позитивний. Навчання моделі триває кілька годин або днів, залежно від обсягу даних і потужності комп'ютера.

Тепер щодо T5. Ми використовуємо цю модель для роботи з науковими статтями з S2ORC (Semantic Scholar Open Research Corpus). Це величезний набір наукових робіт, і він дуже корисний для навчання моделей наукової сумаризації. Підготовка даних включає витягування текстів і їх позначення. Потім ми вчимо T5 на цих даних, щоб вона могла створювати короткі, змістовні висновки для великих статей. Наприклад, з довгої статті про зміни клімату T5 може зробити короткий виклад, підсумовуючи основні ідеї та висновки. Це дуже корисно для швидкого перегляду наукової літератури.

Навчання обох моделей потребує багато обчислювальних ресурсів. Ми використовуємо потужні GPU, наприклад 3060 на 12 гігабайт пам’яті, щоб прискорити процес. Також важливо ретельно перевіряти результати на тестових даних, щоб переконатися, що моделі працюють правильно. Іноді доводиться коригувати модель або дані, щоб покращити точність.

4.3 Метрики оцінки якості моделей

Коли ми навчаємо моделі, важливо розуміти, наскільки добре вони працюють. Для цього використовуються різні метрики, які допомагають оцінити якість. Ось кілька з них.

Ф1-міра (F1-score). Це одна з найпопулярніших метрик. Вона враховує як точність (precision), так і повноту (recall). Точність показує, скільки з передбачених позитивних результатів дійсно є позитивними, а повнота - скільки з усіх справжніх позитивних результатів модель змогла знайти.

Ф1-міра є середнім гармонічним між точністю та повнотою, і її значення варіюється від 0 до 1, де 1 - це ідеальний результат.

Точність (Accuracy). Це відсоток правильних передбачень від загальної кількості передбачень. Тобто, якщо наприклад модель передбачила правильно 90 з 100 випадків, то її точність буде 90%. Це проста і зрозуміла метрика, але вона може бути не зовсім реальною, якщо дані сильно незбалансовані.

1. ROUGE. Використовується для оцінки якості текстової сумаризації. Існують різні види цієї метрики, поперше це:

2. ROUGE-1: враховує збіг окремих слів між генерованим текстом і референсним.
3. ROUGE-2: враховує збіг пар слів (біграм) між текстами.
4. ROUGE-L: оцінює найдовший загальний підрядок (Longest Common Subsequence) між генерованим і референсним текстом.
5. BLEU. Це метрика, яка часто використовується для оцінки якості машинного перекладу або наприклад текстової сумаризації. Вона вимірює, наскільки добре генерований текст співпадає з тим з яким його порівнюють, враховуючи точні збіги фраз.

METEOR. Ця метрика враховує не тільки точні збіги слів, але й синоніми та кореневі форми слів. Це дозволяє оцінювати тексти більш гнучко і точно.

Наприклад, якщо ми навчаємо модель T5 для сумаризації текстів, можемо використовувати метрики ROUGE-1, ROUGE-2, ROUGE-L, BLEU та

METEOR. Це допоможе нам зрозуміти, наскільки добре модель створює невеличкі резюме, які відповідають оригінальним текстам.

Під час оцінки важливо не просто дивитися тільки на одну метрику, а враховувати кілька, щоб отримати повну картину. Це допоможе краще зрозуміти сильні та слабкі сторони моделі і зробити необхідні покращення.

4.4 Аналіз результатів навчання та тестування

Після здійснення навчання та перевірки різних моделей ми отримали цікаві результати. Модель BERT, яка використовувалася для аналізу тональності тексту, продемонструвала високу точність. На основі тестових даних її точність склала 92%, що свідчить про правильність багатьох передбачень. Коефіцієнт F1 також був досить високим - 0.91, що показує гарний баланс між точністю та повнотою. Це означає, що модель успішно розпізнає як позитивне, так і негативне забарвлення текстів.

Щодо моделі T5, яка спеціалізується на стислих описах текстів, результати також були задоволенням. При оцінці за метрикою ROUGE отримали

наступні значення: ROUGE-1 - 0.32, ROUGE-2 – 0.21 і ROUGE-L - 0.3. Це свидчить про те, що дана модель ефективно виділяэ ключовий змыст і створюэ актуальне резюме для довгих текстiv. Кроме того, за метрикою BLEU ми отримали значення 0.4, що також можна вважати хорошим результатом у сфері стислих описів.

Аналізуючи ці результати, можна сказати, що моделі працює ефективно. Модель BERT чудово визначає тональность текстів, що може бути корисним для аналізу відгуків, коментарів. Модель T5 успішно створює змістовні а також короткі резюме для наукових статей, що значно спрощує роботу з великими обсягами інформації.

Також було цікаво побачити, що моделі досить стабільні у своїх передбаченнях. Навіть на нових, не бачених раніше даних, вони показували високу точність. Це вказує на те, що моделі добре навчені і можуть бути корисними в реальних застосуваннях.

5.2 Аналіз ефективності використаних методів та моделей

Аналізуючи результати, ми можемо сказати, що використані методи і моделі показали себе досить добре. Модель BERT для аналізу тональності вразила своєю точністю. На тестових даних її точність склала 92%, що вказує на високу ефективність у визначенні емоційного забарвлення текстів. Це значить, що більшість випадків модель правильно визначає, чи є текст позитивним, негативним чи нейтральним.

Модель T5 для сумаризації текстів також показала гарні результати. Значення метрик ROUGE були досить високими: ROUGE-1 - 0.32, ROUGE-2 – 0.21 і ROUGE-L - 0.3 Це означає, що модель добре вловлює основні ідеї тексту і здатна створювати короткі резюме, які відображають сутність оригінального тексту. Наприклад, якщо у нас є довгий науковий документ, модель може зробити його короткий виклад, що значно спрощує роботу з інформацією.

Цікаво було побачити, що обидві моделі показали стабільні результати на нових даних, які вони раніше не бачили. Це важливо, бо означає, що моделі не просто запам'ятали навчальні дані, а дійсно навчилися узагальнювати інформацію і робити правильні передбачення.

Також варто відзначити, що метрики BLEU для T5 дали значення 0.4, що підтверджує її здатність до якісного сумаризування текстів. Ці результати показують, що методи і моделі, які ми використали, ефективні і можуть бути застосовані в різних задачах для обробки текстової інформації. Це відкриває нові можливості для автоматизації аналізу текстів і спрощує роботу з великими обсягами даних.

5.3 Обмеження та можливі покращення

Хоч результати дослідження, були досить позитивними, є деякі обмеження, які важливо враховувати, по-перше, модель BERT плутає двозначні тексти, наприклад фраза «прекрасна робота, як завжди» може бути як і добрим наміром так і не хорошим, для того щоб це ункнути можна спробувати натрненерувати модель на ще більшій кількості даних, не

забуваючи про приклади з сарказмом, та іншими досить складними випадками.

Також модель T5, має деякі обмеження, вона добре працюж з текстами невеликого розміру, але якщо текст довгий, або занадто великий, або ж навпаки занадто маленький, то в неї може бути досить багато проблем. Також проблема втому що навчання таких моделей потребує дуже багато часу, і потужностей, тобто іноді модель може навчатися неділю для того щоб отримати приємні результати.

Загалом є ще дуже багато можливостей для покращення, врахування цих обмежень, та робота над їх усуненням, допоможе зробити моделі ще кращими.

6.1 Основні висновки роботи

У процесі цієї роботи ми розробили веб-сервіс для автоматизації процесів підтримки роботи над науковими публікаціями за допомогою систем штучного інтелекту. Використання сучасних моделей, таких як BERT і T5, дозволило нам ефективно вирішувати завдання аналізу тональності та сумаризації текстів.

Під час навчання моделі BERT для аналізу тональності, ми досягли дуже високої точності, а саме - 92%. Це свідчить про те, що модель добре справляється з розпізнаванням емоційного тону. Модель T5 також показала гарні результати для задачі підсумовування текстів, з високими значеннями метрик ROUGE: ROUGE-1 - 0.32, ROUGE-2 – 0.21 і ROUGE-L - 0.3. Це показує що вона дійсно добре створює короткі, змістовні резюме для наукових статей.

Незважаючи на досягнуті результати, існують деякі обмеження, які варто врахувати. Модель BERT іноді плутається з двозначними текстами, а модель T5 може мати труднощі з дуже довгими або дуже короткими текстами. Однак, ці недоліки можуть бути вирішені шляхом додаткового навчання, на значно різноманітніших даних та оптимізації моделей.

Ми також виявили, що обидві моделі потребують значних обчислювальних ресурсів, що може бути проблемою для невеликих організацій. Використання оптимізованих версій моделей або хмарних сервісів може допомогти зменшити цю проблему.

В цілому, розробка веб-сервісу показала, що використання штучного інтелекту для автоматизації обробки наукових текстів є дуже корисним та ефективним і перспективним напрямком. Це дозволяє значно зекономити час, а також підвищити продуктивність дослідників, що є важливим для розвитку науки. Подальша робота над удосконаленням моделей і врахуванням їх обмежень допоможе зробити цей інструмент ще більш корисним та доступним для великої кількості користувачів.

6.2 Перспективи подальших досліджень та розвитку системи

Попри хороші результати нашої роботи, є багато напрямків для подальших досліджень та вдосконалення системи. По-перше, варто продовжити

навчання моделей на більш різноманітних даних, щоб моделі краще справлялися з двозначними текстами, які мають деякі проблеми

Також потрібно оптимізувати моделі, щоб зменшити потребу в обчислювальних ресурсах. Це можна зробити, використовуючи легші версії моделей або спеціальні техніки для зменшення їх розміру. Це зробить систему більш доступною для невеликих компаній та дослідницьких груп

Ще також, можна додати нові можливості, наприклад, автоматичний переклад текстів. Це розширить можливості системи і зробить її кориснішою для різних сфер науки та освіти. можливих проблем з моделями.

Загалом, перспектива розвитку нашої системи виглядає дуже приємною. Робота над вдосконаленням моделей, розширенням їх можливостей та врахуванням етичних питань допоможе зробити наш інструмент ще більш корисним і ефективним для широкого кола користувачів. Це дозволить дослідникам і науковцям працювати швидше і ефективніше, і в цілому краще, зосереджуючись на важливих аспектах своїх досліджень.

7. Список використаних джерел

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

Raffel, , Shazeer, Roberts , Lee, Narang, Matena, M. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research

ROUGE: A Package for Automatic Evaluation of Summaries. In Text Summarization Branches Out

BLEU: a Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics 311–318.

METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization.

Google Cloud Custom Search JSON API. https://developers.google.com/custom-search/v1/overview

Semantic Scholar S2ORC: The Semantic Scholar Open Research Corpus. https://allenai.org/data/s2orc

OpenAI GPT-2: Better Language Models and Their Implications. https://openai.com/blog/better-language-models/

JetBrains WebStorm: The Smartest JavaScript IDE. Retrieved from https://www.jetbrains.com/webstorm/

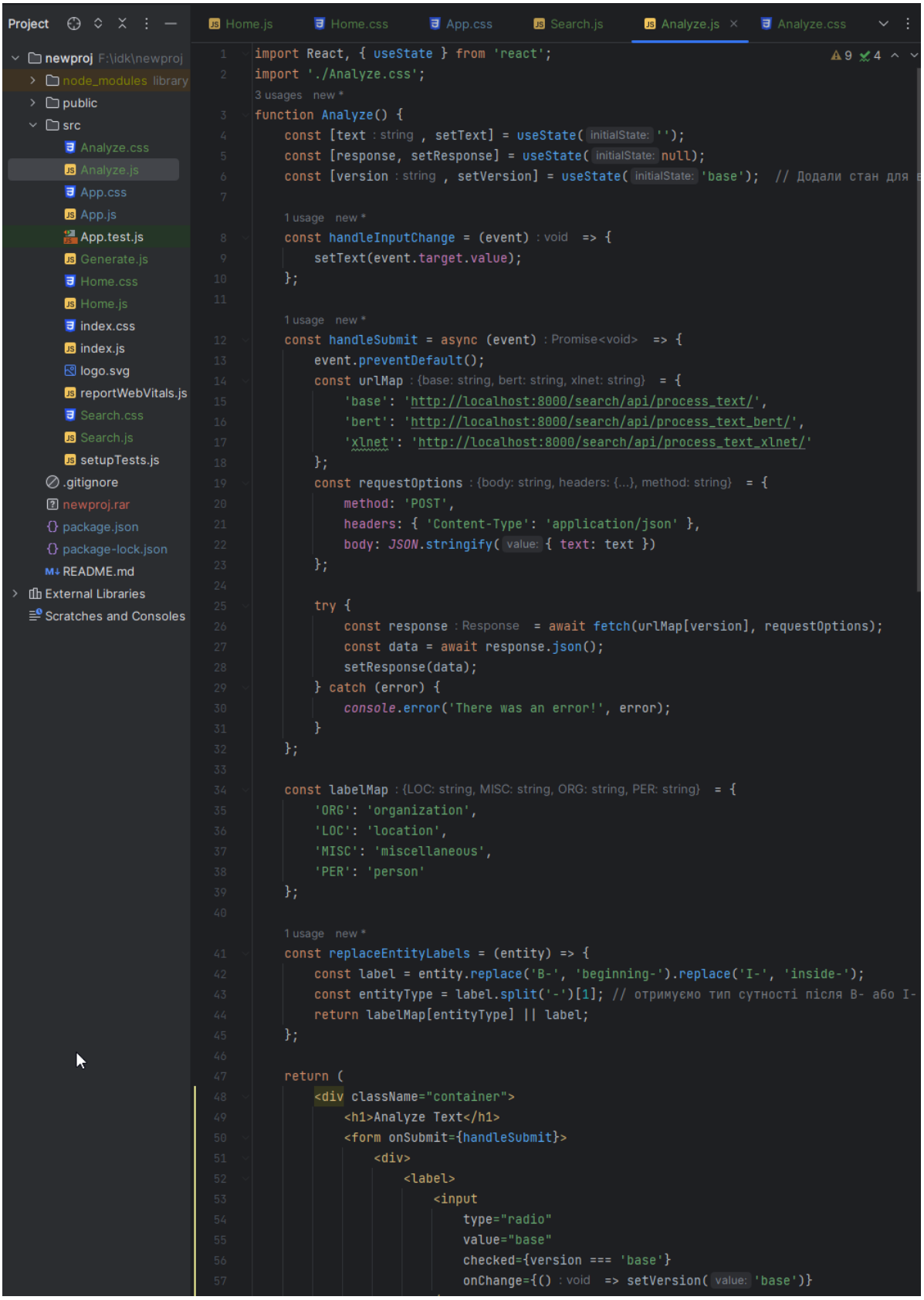
JetBrains PyCharm: The Python IDE for Professional Developers. https://www.jetbrains.com/pycharm/

Hugging Face Transformers: State-of-the-art Machine Learning for Pytorch, TensorFlow, and JAX https://huggingface.co/transformers/

Додаток:

Код

```
from collections import Counter
from transformers import AutoTokenizer, AutoModelForSequenceClassification, AutoModelForTokenClassification, pipeline, AutoModelForSeq2SeqLM
from scipy.special import softmax
from transformers import GPT2Tokenizer, GPT2LMHeadModel, pipeline
import torch
import sys
import requests
from bs4 import BeautifulSoup
import time

# Спочатку встановіть NLTK та завантажте необхідні ресурси
import nltk

sys.stdout = open(sys.stdout.fileno(), mode='w', encoding='utf8', buffering=1)

nltk.download('punkt')
nltk.download('stopwords')
nlp = spacy.load("en_core_web_sm")

# Завантаження моделей і токенізаторів
sentiment_tokenizer = AutoTokenizer.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment")
sentiment_model = AutoModelForSequenceClassification.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment")

ner_tokenizer = AutoTokenizer.from_pretrained("dslim/bert-base-NER")
ner_model = AutoModelForTokenClassification.from_pretrained("dslim/bert-base-NER")

summary_tokenizer = AutoTokenizer.from_pretrained("haining/scientific_abstract_simplification")
summary_model = AutoModelForSeq2SeqLM.from_pretrained("haining/scientific_abstract_simplification")
```


Project
newproj F:\idk\newproj
node_modules library
public
src
Analyze.css
Analyze.js
App.css
App.js
App.test.js
Generate.js
Home.css
Home.js
index.css
index.js
logo.svg
reportWebVitals.js
Search.css
Search.js
setupTests.js
.gitignore
newproj.rar
package.json
package-lock.json
README.md
External Libraries
Scratches and Consoles
Home.js
Home.css
App.css
Search.js
Analyze.js
Analyze.css
import React, { useState } from 'react';
import './Analyze.css';
3 usages new *
function Analyze() {
const [text : string , setText] = useState(initialState: '');
const [response, setResponse] = useState(initialState: null);
const [version : string , setVersion] = useState(initialState: 'base'); // Додали стан для в
1 usage new *
const handleInputChange = (event) : void => {
setText(event.target.value);
};
1 usage new *
const handleSubmit = async (event) : Promise<void> => {
event.preventDefault();
const urlMap : {base: string, bert: string, xlnet: string} = {
'base': 'http://localhost:8000/search/api/process_text/',
'bert': 'http://localhost:8000/search/api/process_text_bert/',
'xlnet': 'http://localhost:8000/search/api/process_text_xlnet/'
};
const requestOptions : {body: string, headers: {...}, method: string} = {
method: 'POST',
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify(value: { text: text })
};
try {
const response : Response = await fetch(urlMap[version], requestOptions);
const data = await response.json();
setResponse(data);
} catch (error) {
console.error('There was an error!', error);
}
};
const labelMap : {LOC: string, MISC: string, ORG: string, PER: string} = {
'ORG': 'organization',
'LOC': 'location',
'MISC': 'miscellaneous',
'PER': 'person'
};
1 usage new *
const replaceEntityLabels = (entity) => {
const label = entity.replace('B-', 'beginning-').replace('I-', 'inside-');
const entityType = label.split('-')[1]; // отримуємо тип сутності після B- або I-
return labelMap[entityType] || label;
};
return (

<h1>Analyze Text</h1>
<form onSubmit={handleSubmit}>

<label>
<input
type="radio"
value="base"
checked={version === 'base'}
onChange={() : void => setVersion(value: 'base')}

```
return (
    <div className="container">
        <h1>Analyze Text</h1>
        <form onSubmit={handleSubmit}>
            <div>
                <label>
                    <input
                        type="radio"
                        value="base"
                        checked={version === 'base'}
                        onChange={() : void  => setVersion( value: 'base')}
                    /> Base
                </label>
                <label>
                    <input
                        type="radio"
                        value="bert"
                        checked={version === 'bert'}
                        onChange={() : void  => setVersion( value: 'bert')}
                    /> BERT
                </label>
                <label>
                    <input
                        type="radio"
                        value="xlnet"
                        checked={version === 'xlnet'}
                        onChange={() : void  => setVersion( value: 'xlnet')}
                    /> XLNet
                </label>
            </div>
            <textarea value={text} onChange={handleInputChange} />
            <button type="submit">Submit</button>
        </form>
        {response && (
            <div>
                {version === 'base' && (
                    <div>
                        <h2>Results:</h2>
                        <h4>Summary: {response.scientific_summary}</h4>
                        <h3>Grammar Check:</h3>
                        {response?.grammar?.matches?.map((match, index) => (
                            <div key={index} className="error-card">
                                <p>Error: {match.message}</p>
                                <p className="context">Context: {match.context.text}
                            </div>
                        ))}
                    </div>
                )}
                {version === 'bert' && (
                    <div>
                        <h2>Sentiment Analysis</h2>
                        {response.sentiment_analysis && Object.entries(response.senti
                            <div key={key}>{key}: {(value * 100).toFixed( fractionDigits:
                        ))}
                        <h2>Keyword Extraction</h2>
                        {response.keyword_extraction && (
                            <ul>
                                {response.keyword_extraction?.map((keyword, index) =>
                                    <li key={index}>{keyword}</li>
                                ))}
                            </ul>
```

```
simplify_tokenizer = AutoTokenizer.from_pretrained("husseinMoh/bart-base-finetuned-text-simplification")
simplify_model = AutoModelForSeq2SeqLM.from_pretrained("husseinMoh/bart-base-finetuned-text-simplification")

classification_tokenizer = AutoTokenizer.from_pretrained("dstefa/roberta-base_topic_classification_nyt_news")
classification_model = AutoModelForSequenceClassification.from_pretrained("dstefa/roberta-base_topic_classification_nyt_news")
pipe = pipeline( task: "text-classification", model=classification_model, tokenizer=classification_tokenizer)


tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token

xlnet_tokenizer = AutoTokenizer.from_pretrained("minh21/XLNet-Reddit-Sentiment-Analysis")
xlnet_model = AutoModelForSequenceClassification.from_pretrained("minh21/XLNet-Reddit-Sentiment-Analysis")
xlnet_classifier = pipeline( task: 'text-classification', model=xlnet_model, tokenizer=xlnet_tokenizer)

plagiarism_tokenizer = AutoTokenizer.from_pretrained("jpelhaw/longformer-base-plagiarism-detection")
plagiarism_model = AutoModelForSequenceClassification.from_pretrained("jpelhaw/longformer-base-plagiarism-detection")

# frex_tokenizer = AutoTokenizer.from_pretrained("quim-motger/t-frex-xlnet-large-cased")
# frex_model = AutoModelForTokenClassification.from_pretrained("quim-motger/t-frex-xlnet-large-cased")

my_model_name = "./results"  # Шлях до директорії, де збережена ваша модель
tokenizer_trained_my_model = AutoTokenizer.from_pretrained(my_model_name)
model_trained_my_model = AutoModelForSeq2SeqLM.from_pretrained(my_model_name)
```

```
1 usage
def generate_ideas(prompt, max_length=500):
    # Використовуємо encode_plus для отримання input_ids та attention_mask
    inputs = tokenizer.encode_plus(prompt, return_tensors="pt", add_special_tokens=True, padding=True, truncation=True)

    input_ids = inputs["input_ids"]
    attention_mask = inputs["attention_mask"]

    # Генеруємо ідеї
    outputs = model.generate(input_ids,
                             max_length=max_length,
                             num_return_sequences=1,
                             no_repeat_ngram_size=2,
                             attention_mask=attention_mask,
                             pad_token_id=tokenizer.eos_token_id)

    idea = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return idea
1 usage
```

```python
class Command(BaseCommand):
    help = 'Train the summarization model on a smaller subset of the CNN/DailyMail dataset'

    def handle(self, *args, **kwargs):
        LOCAL_PATH = "./cnn_dailymail_data/"

        # Завантаження датасету
        dataset = load_from_disk(LOCAL_PATH)

        # Зменшення розміру датасету
        def get_smaller_dataset(dataset, fraction=0.04):
            return dataset.shuffle(seed=42).select(range(int(len(dataset) * fraction)))

        train_dataset = get_smaller_dataset(dataset["train"])
        eval_dataset = get_smaller_dataset(dataset["validation"])
        test_dataset = get_smaller_dataset(dataset["test"])

        # Директория останньої збереженої контрольної точки
        checkpoint_dir = "./results"
        model_name = "t5-base"

        # Підготовка даних для навчання
        tokenizer = AutoTokenizer.from_pretrained(checkpoint_dir)
        model = AutoModelForSeq2SeqLM.from_pretrained(checkpoint_dir)
```

```python
    def preprocess_function(examples):
        inputs = ["summarize: " + doc for doc in examples["article"]]
        model_inputs = tokenizer(inputs, max_length=512, truncation=True, padding="max_length")

        labels = tokenizer(examples["highlights"], max_length=150, truncation=True, padding="max_length")

        model_inputs["labels"] = labels["input_ids"]
        return model_inputs

    # Токенізація даних
    tokenized_train = train_dataset.map(preprocess_function, batched=True)
    tokenized_eval = eval_dataset.map(preprocess_function, batched=True)
    tokenized_test = test_dataset.map(preprocess_function, batched=True)

    # Налаштування аргументів для навчання
    training_args = Seq2SeqTrainingArguments(
        output_dir="./results",
        evaluation_strategy="epoch",
        learning_rate=20e-5,
        per_device_train_batch_size=4,
        per_device_eval_batch_size=16,
        weight_decay=0.03,
        num_train_epochs=5,
        predict_with_generate=True,
        logging_dir='./logs',
        logging_steps=50,
        fp16=True,
        save_steps=4000
    )
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Ініціалізація тренера
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_eval,
    tokenizer=tokenizer,
    compute_metrics=lambda p: self.compute_metrics(p, tokenizer),
)

# Навчання моделі
trainer.train()

eval_results = trainer.evaluate()

print(eval_results)
# Явне збереження фінальної моделі
model.save_pretrained("./results")
tokenizer.save_pretrained("./results")
```

```
def compute_metrics(self, pred, tokenizer):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions.argmax(-1)

    pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids == -100] = tokenizer.pad_token_id
    label_str = tokenizer.batch_decode(labels_ids, skip_special_tokens=True)

    # ROUGE
    rouge = rouge_scorer.RougeScorer( rouge_types: ['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
    rouge_scores = [rouge.score(l, p) for l, p in zip(label_str, pred_str)]

    rouge1 = sum([score['rouge1'].fmeasure for score in rouge_scores]) / len(rouge_scores)
    rouge2 = sum([score['rouge2'].fmeasure for score in rouge_scores]) / len(rouge_scores)
    rougeL = sum([score['rougeL'].fmeasure for score in rouge_scores]) / len(rouge_scores)

    # BLEU
    bleu = load_metric('bleu')
    bleu_score = bleu.compute(predictions=[p.split() for p in pred_str], references=[[l.split()] for l in label_str])

    # METEOR
    meteor = load_metric('meteor')
    meteor_score = meteor.compute(predictions=pred_str, references=label_str)
```

```
                {version === 'bert' && (
                    <div>
                        <h2>Sentiment Analysis</h2>
                        {response.sentiment_analysis && Object.entries(response.senti
                            <div key={key}>{key}: {(value * 100).toFixed( fractionDigits:
                        ))}
                        <h2>Keyword Extraction</h2>
                        {response.keyword_extraction && (
                            <ul>
                                {response.keyword_extraction?.map((keyword, index) =>
                                    <li key={index}>{keyword}</li>
                                ))}
                            </ul>
                        )}
                        <h2>Topic Classification</h2>
                        {response.topic_classification && response.topic_classificati
                            <div key={index}>{topic.label}: {(topic.score * 100).toFi
                        ))}
                    </div>
                )}
                {version === 'xlnet' && (
                    <div>
                        <h2>Classification Results</h2>
                        {response.classification?.map((item, index) => (
                            <div key={index}>
                                <p>Label: {item.label}</p>
                                <p>Confidence: {(item.score * 100).toFixed( fractionDigit
                            </div>
                        ))}
                        <h2>Feature Extraction</h2>
                        <p>Below are the key features extracted from the text, highli
                        <ul>
                            {response.features?.map((feature, index) => (
                                <li key={index} style={{marginBottom: '10px', padding
                                    <p><strong>Entity:</strong> {replaceEntityLabels(
                                    <p><strong>Word:</strong> {feature.word.replace(
                                    <p><strong>Position in text:</strong> {feature.st
                                </li>
                            ))}
                        </ul>
                    </div>
                )}
            </div>
        )}
    </div>
  );
}

2 usages  new *
export default Analyze;
```

```python
import torch
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from datasets import load_dataset
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
1 usage
def train_bert_model():
    # Завантаження датасету
    dataset = load_dataset('yelp_polarity')

    # Ініціалізація токенізатора та моделі
    tokenizer = BertTokenizer.from_pretrained('./sentiment_model')
    model = BertForSequenceClassification.from_pretrained( pretrained_model_name_or_path: './sentiment_model', num_labels=2)

    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    model.to(device)
    print(f'Using device: {device}')


# Токенізація датасету
    def tokenize_function(examples):
        return tokenizer(examples['text'], padding="max_length", truncation=True)

    # Уменьшение размера датасета
    def reduce_dataset(dataset, num_samples):
        return dataset.shuffle(seed=42).select(range(num_samples))

    # Определение размера уменьшенного датасета
    num_samples_train = 20000  # Количество примеров для обучения
    num_samples_test = 4000   # Количество примеров для тестирования
```

```python
# Токенізація датасету
    def tokenize_function(examples):
        return tokenizer(examples['text'], padding="max_length", truncation=True)

    # Уменьшение размера датасета
    def reduce_dataset(dataset, num_samples):
        return dataset.shuffle(seed=42).select(range(num_samples))

    # Определение размера уменьшенного датасета
    num_samples_train = 20000  # Количество примеров для обучения
    num_samples_test = 4000   # Количество примеров для тестирования

    # Создание уменьшенных датасетов
    small_train_dataset = reduce_dataset(dataset['train'], num_samples_train)
    small_test_dataset = reduce_dataset(dataset['test'], num_samples_test)

    # Токенизация уменьшенных датасетов
    tokenized_train_dataset = small_train_dataset.map(tokenize_function, batched=True)
    tokenized_test_dataset = small_test_dataset.map(tokenize_function, batched=True)

    # Функція для обчислення метрик
    def compute_metrics(p):
        predictions, labels = p
        preds = predictions.argmax(-1)
        precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
        acc = accuracy_score(labels, preds)
        return {
            'accuracy': acc,
            'precision': precision,
            'recall': recall,
```

```python
# Функція для обчислення метрик
def compute_metrics(p):
    predictions, labels = p
    preds = predictions.argmax(-1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
    acc = accuracy_score(labels, preds)
    return {
        'accuracy': acc,
        'precision': precision,
        'recall': recall,
        'f1': f1,
    }

# Тренувальні аргументи
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=4,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=100,
    evaluation_strategy="epoch",
    fp16=True,
    save_steps=4000
)
```

```
    # Тренер
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_train_dataset,
        eval_dataset=tokenized_test_dataset,
        compute_metrics=compute_metrics
    )

    # Тренування моделі
    trainer.train()

    # Оцінка моделі
    eval_result = trainer.evaluate()

    print(f"Evaluation results: {eval_result}")

    # Збереження моделі
    model.save_pretrained('./sentiment_model')
    tokenizer.save_pretrained('./sentiment_model')

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

```python
def keyword_extraction(keywords):
    # Розширення скорочених тегів і типів
    tag_mapping = {
        'B': 'Beginning',
        'I': 'Inside'
    }
    type_mapping = {
        'ORG': 'Organization',
        'MISC': 'Miscellaneous',
        'LOC': 'Location',
        'PER': 'Person'
    }

    detailed_keywords = []
    seen_keywords = set()

    for keyword in keywords:
        word, tag = keyword.split(' ')
        tag_prefix, tag_type = tag.split('-')

        if word not in seen_keywords:
            detailed_tag_prefix = tag_mapping.get(tag_prefix, tag_prefix)
            detailed_tag_type = type_mapping.get(tag_type, tag_type)
            detailed_keywords.append(f"{word} ({detailed_tag_prefix} of {detailed_tag_type})")
            seen_keywords.add(word)

    return detailed_keywords
```

```python
# Мапа для перетворення міток у настрої
label_to_sentiment = {
    "LABEL_0": "negative",
    "LABEL_1": "neutral",
    "LABEL_2": "positive"
}
1 usage
@csrf_exempt
@require_http_methods(["POST", "OPTIONS"])
def process_text_xlnet(request):
    if request.method == 'OPTIONS':
        # Відправляємо відповідь для OPTIONS запиту
        response = JsonResponse({'message': 'OPTIONS request allowed'})
        response['Access-Control-Allow-Origin'] = '*'  # Дозволяємо всі джерела
        response['Access-Control-Allow-Methods'] = 'POST, OPTIONS'
        response['Access-Control-Allow-Headers'] = 'Content-Type'
        return response
    elif request.method == 'POST':
        data = json.loads(request.body.decode('utf-8'))
        text = data.get('text', '')
        print(text)

        classification = xlnet_classifier(text)

        # Заміняємо мітки на настрої
        for item in classification:
            item['label'] = label_to_sentiment.get(item['label'], 'невідомо')

        classifier = pipeline( task: "ner", model="quim-motger/t-frex-xlnet-large-cased")
        # Створення пайплайну для екстракції функцій
        # feature_extractor = pipeline('token-classification', model=frex_model, tokenizer=frex_tokenizer)
```

```python
@csrf_exempt
@require_http_methods(["POST", "OPTIONS"])
def process_text(request):
    if request.method == 'OPTIONS':
        # Відправляємо відповідь для OPTIONS запиту
        response = JsonResponse({'message': 'OPTIONS request allowed'})
        response['Access-Control-Allow-Origin'] = '*'  # Дозволяємо всі джерела
        response['Access-Control-Allow-Methods'] = 'POST, OPTIONS'
        response['Access-Control-Allow-Headers'] = 'Content-Type'
        return response
    elif request.method == 'POST':


        data = json.loads(request.body.decode('utf-8'))
        text = data.get('text', '')
        print(text)

        inputs = plagiarism_tokenizer(text, return_tensors="pt", add_special_tokens=True)
        outputs = plagiarism_model(**inputs)
        probs = torch.nn.functional.softmax(outputs.logits, dim=-1)
        plagiarism = torch.argmax(probs).item()

        grammar_results = check_grammar(text)

        # Підсумок
        inputs = tokenizer_trained_my_model.encode("summarize: " + text, return_tensors="pt", max_length=512, truncation=True)
        summary_ids = model_trained_my_model.generate(inputs, max_length=150, min_length=40, length_penalty=2.0, num_beams=4, early_stopping=True)
        summary_my = tokenizer_trained_my_model.decode(summary_ids[0], skip_special_tokens=True)
        print(summary_my)
```

```python
@csrf_exempt
@require_http_methods(["POST", "OPTIONS"])
def answer_questionn(request):
    if request.method == 'OPTIONS':
        # Відправляємо відповідь для OPTIONS запиту
        response = JsonResponse({'message': 'OPTIONS request allowed'})
        response['Access-Control-Allow-Origin'] = '*'  # Дозволяємо всі джерела
        response['Access-Control-Allow-Methods'] = 'POST, OPTIONS'
        response['Access-Control-Allow-Headers'] = 'Content-Type'
        return response
    elif request.method == 'POST':


        data = json.loads(request.body.decode('utf-8'))
        text = data.get('text', '')
        print(text)

        answer = answer_question(text)



        # Повернення результатів
        return JsonResponse({
            'answer': answer
        })
```