МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Львівський національний університет імені Івана Франка

Факультет електроніки та комп'ютерних технологій

Кафедра системного проектування

Допустити до захисту
Завідувач кафедри
_____доц. Шувар Р.Я.
«___» _____2023 р.

Кваліфікаційна робота

Бакалавр
(освітній ступінь)

Побудова системи запитання-відповіді на основі нейронної мережі BERT

Виконав:
студент групи ФЕП – 41
спеціальності 121 – Інженерія
програмного забезпечення

Лещенко М. О.
Науковий керівник:
ас. Парубочий В.О.
«____» _____2023 р.

Рецензент:
доц. Карбовник І.Д.

Львів 2023

АНОТАЦІЯ

Дана бакалаврська робота має за мету розробку системи запитань та відповідей на основі наданого контексту. Система побудована на нейронній мережі BERT, яка натренована на наборі даних, перекладених українською мовою, що покращує точність відповідей. Основна ідея полягає у створенні системи, що здатна чітко та правильно формулювати відповіді на запитання у визначеній галузі знань. Для зручності використання ця система має вебінтерфейс, де користувач має змогу завантажити документ або ж текст, та почати ставити запитання за його контекстом. Сервісом також можна користуватись через API, що дає змогу компаніям застосовувати його на власних платформах та значно розширює спектр його використання.

Під час роботи над цією проблемою була розглянута велика кількість технологій, що дало змогу побудувати гнучку систему, здатну до швидкого масштабування. Значну частину роботи склали пошук та вибір архітектури нейронної мережі, що дозволило обрати найбільш дієву та точну модель для нашого завдання. Також не менш важливою частиною роботи були пошук та обробка набору даних, застосування якого дозволило отримати найвищі результати в ході виконання поставленого завдання. Використання хмарних обчислень допомогло досягти бажаного результату за короткі терміни, а також зробило можливим використання мікросервісної архітектури для нашого додатка, обгорнувши нашу модель в API сервіс.

Результати даної роботи можуть бути використаними для вирішення різного роду завдань. Зокрема, компанії можуть користуватись цим сервісом, щоб підвищити швидкість та точність відповідей на найпоширеніші запитання їхніх користувачів. Також платформа може відігравати роль пошукового сервісу, де для того, щоб здійснити пошук інформації за документом, користувачу не обов’язково знати точне слово. Представлена модель здатна розуміти запитання,

сформульовані природною мовою, та загальний контекст поставленого питання, щоб дати змістовну, чітку відповідь.

Загалом, розвиток та вдосконалення системи запитань та відповідей дають нам можливість створення потужного та надійного інструменту, що зможе забезпечити користувачам швидкий та зручний пошук інформації у великих масивах даних.

ABSTRACT

The aim of this bachelor's thesis is to develop a question-and-answer system based on the provided context. The system is built on a BERT neural network and fine-tuned on a Ukrainian dataset, which enhances the accuracy of the answers. The main objective is to create a system capable of formulating clear and correct answers to questions in a specific domain of knowledge. The system has a web-based interface for user-friendly access, allowing users to upload documents or texts and start posing questions related to the content. Additionally, the service can be utilized through an API, enabling companies to integrate it into their own platforms and greatly expanding its range of applications.

During the course of this project, a range of technologies was considered, resulting in the development of a flexible system with the ability for rapid scalability. A significant portion of the effort was dedicated to searching and selecting the neural network architecture, which facilitated the identification of the most efficient and accurate model for the task at hand. Equally important aspects of the project involved searching for and processing the dataset, which was subsequently used to achieve the best possible model performance for the given task. The utilization of cloud computing expedited the attainment of the desired outcome within a short timeframe, while also enabling the adoption of a microservice architecture by wrapping our model in an API service.

The outcomes of this work can be applied to solve various types of problems. Business owners can leverage this service to enhance the speed and accuracy of responses to frequently asked questions. Moreover, the platform can be employed as a search service, allowing users to retrieve information without needing to know the precise keywords. The model comprehends natural language questions and context, enabling it to locate answers that may even contain synonyms found within the text.

Overall, further research and development of the Q&A system can yield a robust and dependable tool that facilitates prompt and accurate processing of user queries. This will contribute to enhanced service quality and improved usability of information resources.

ЗМІСТ

ЗМІСТ *4*

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ *6*

ВСТУП *11*

1 АНАЛІЗ СТАНУ ПРОБЛЕМНОЇ ОБЛАСТІ *13*

1.1 Історія виникнення **13**

1.2 Відмінності в архітектурі **14**

1.3 Парадигми системвідповідей на запитання **15**

1.4 Виклики під час надання відповідей на запитання **16**

2 ВИКОРИСТАННЯ НЕЙРОННИХ МОДЕЛЕЙ ДЛЯ ПОБУДОВИ СИСТЕМИ ЗАПИТАНЬ ТА ВІДПОВІДЕЙ *18*

2.1 Сучасні системи запитань та відповідей та AI **18**

2.2 Архітектура Transformer **18**

2.3 Механізм уваги **21**

2.4 Multi-head Attention **23**

2.5 Позиційне кодування (Positional Encoding) **24**

2.6 BERT (Bidirectional Encoder Representations from Transformers) **25**

2.6.2 Моделювання замаскованої мови (Masked LM) **27**

2.6.3 Передбачення наступного речення (NSP) **28**

2.6.4 Донавчання BERT (Fine-tuning) **29**

2.7 Види моделей BERT **31**

2.7.1 XLNet **32**

2.7.2 RoBERTa **33**

2.7.3 DistilBERT **34**

2.7.4 XLM-RoBERTa **34**

2.11 Відмінності RoBERTa від BERT **35**

3. ДАНІ ДЛЯ ТРЕНУВАННЯ СИСТЕМИ ЗАПИТАНЬ ТА ВІДПОВІДЕЙ *41*

3.1 Підбір набору даних **41**

3.2 SQuAD 2.0 (Stanford Question Answering Dataset) перекладений на українську **42**

3.3 Важливість SQuAD 2.0 ..... **43**

3.4 Методи попередньої обробки та архітектури моделей **43**

3.5 Метрики оцінювання та аналіз продуктивності 44
4. ВИКОРИСТАННЯ ХМАРНИХ ОБЧИСЛЕНЬ ДЛЯ ТРЕНУВАННЯ МОДЕЛЕЙ 46
4.1 Amazon Web Services 46
4.2 AWS Lambda 46
4.3 Amazon S3 48
4.4 Amazon API Gateway 48
4.5 Додаткові AWS сервіси для тренування нейронних моделей 49
4.6 Amazon SageMaker 50
5. ПРАКТИЧНА ЧАСТИНА 51
5.1 Розробка та тренування моделі 51
5.2 Модель як API сервіс 62
5.3 Створення вебзастосунку 67
ВИСНОВОК 72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 74
ДОДАТОК А 77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) - це механізми, які дають змогу двом програмним компонентам взаємодіяти один з одним, використовуючи набір визначень і протоколів.

BERT (Bidirectional Encoder Representations from Transformers) - це попередньо навчена модель обробки природної мови, розроблена компанією Google.

NLP (Natural Language Processing) - підгалузь штучного інтелекту, зосереджена на взаємодії між комп'ютером і людською мовою.

QA system (Question and Answering system) – це комп'ютерна програма або модель, призначена для розуміння запитань, поставлених природною мовою, та відповідей на ці запитання.

RNN (Recurrent Neural Network) - це рекурентна нейронна мережа, призначена для обробки послідовних даних.

SQL (Structured Query Language - мова структурованих запитів) і SPARQL (SPARQL Protocol and RDF Query Language - мова запитів) - це мови запитів, які використовуються в різних контекстах. SQL в основному використовується для управління та запитів до реляційних баз даних. Вона дає змогу користувачам визначати, маніпулювати та отримувати структуровані дані, що зберігаються в реляційних системах управління базами даних (СКБД).

ВСТУП

Відповіді на запитання - важлива проблема NLP і давня віха в розвитку штучного інтелекту. Системи запитань та відповідей дають змогу користувачеві поставити запитання природною мовою та отримати миттєву і чітку відповідь. Зараз системи запитань та відповідей можна знайти в пошукових системах і телефонних діалогових інтерфейсах, і вони досить добре відповідають на прості фрагменти інформації. Однак, попри широке використання цієї технології в різних рішеннях, її реалізація залишається складною, і тому вона досі рідко застосовується в продуктах компаній.

Однією з головних цілей даної роботи є дослідження можливостей нейронних мереж у задачі пошуку відповідей з контексту. Для досягнення цієї мети розробляється вебзастосунок, який мати в собі розробку вебінтерфейс додатка.

Об'єктом дослідження є сам процес розробки вебзастосунку та його компонентів, які забезпечують взаємодію з користувачем і пошук відповідей з наданого контексту.

Методи дослідження включають аналіз інших систем запитань та відповідей, розробку нейронної мережі для пошуку відповідей, використання різних технологій для розробки вебзастосунків.

Важливою відмінністю цієї системи від інших є те, що модель, яка використовується у вебдодатку, є попередньо натренованою на українському наборі даних, що дає змогу отримувати більш точні результати, якщо контекст та питання побудовані українською мовою. На цей момент, на ринку майже неможливо знайти додаток зі схожим принципом, який би був точним у відповідях не тільки в англійській, а і в українській мові.

Одним з наступних етапів розвитку цього продукту може стати здатність обробляти цілі документи, що дозволить користувачеві "переписуватися" зі статтею, книгою та іншими текстовими матеріалами. Ця опція здатна розширити сферу застосування даної системи, та змогла б претендувати на роль повноцінного розумного пошукового сервісу.

Також, використання генеративних нейронних мереж для виконання схожих завдань може покращити отримуваний результат. Використання таких генеративних моделей може привести до більш природних та контекстуальних відповідей, що зробить систему ще більш потужною та корисною

1 АНАЛІЗ СТАНУ ПРОБЛЕМНОЇ ОБЛАСТІ

Розробка систем "запитання-відповідь" (QA) вже понад півстоліття перебуває в центрі уваги в галузі обробки природної мови (NLP). На ранніх стадіях ці системи розроблялися на базових комп'ютерах і в основному були спрямовані на пошук інформації в обмеженій предметній області. Як зазначалося в першому дослідженні систем запитань відповідей, області застосування цих ранніх програм були досить обмеженими, що дозволяло розробникам не помічати багатьох проблем, пов'язаних з розумінням і адекватною відповіддю на запитання.

1.1 Історія виникнення

Перша визнана система запитань відповідей під назвою BASEBALL була представлена в 1961 році Гріном та ін. [10] Вона була розроблена для відповідей на прості запитання про бейсбольні матчі Американської ліги, такі як місяць, місце, команда або рахунок за певний рік гри. Примітно, що BASEBALL покладався на перфокарти для зчитування запитань, що підкреслює зародковий стан комп'ютерних технологій на той час. Основна мета цього дослідження полягала в тому, щоб продемонструвати базовий доказ концепції систем запитань та відповідей.

Подальший розвиток синтаксичних і семантичних аналізаторів призвів до появи наступного покоління систем запитань та відповідей, які дозволили створювати більш виразні вхідні запитання[2]. Однією з найвідоміших систем цієї епохи була LUNAR, розроблена для місячних геологів під час місії "Аполлон". LUNAR дозволила їм отримати доступ і оцінити дані хімічного аналізу складу місячних порід і ґрунту.

З роками системи запитань та відповідей досягли значного прогресу і перевершили попередні еталони здатності відповідати на запитання. Помітною віхою стало створення Watson компанією IBM Research [3], системи запитань та відповідей на основі глибокого навчання, яка брала участь у вікторині "Jeopardy" і перемогла в ній найкращих учасників. Це досягнення стало гордістю для галузі, оскільки продемонструвало здатність систем запитань та відповідей перевершити людський інтелект у ключовому тесті - відповіді на запитання[2].

Оскільки сфера систем запитань та відповідей продовжує розвиватися, постійні дослідження і розробки спрямовані на подальше розширення їхніх можливостей і розв'язання складніших проблем у розумінні природної мови та пошуку інформації.

1.2 Відмінності в архітектурі

Одним з ключових факторів, що відрізняє системи запитань та відповідей, є те, чи призначені вони для додатків з відкритим або закритим доменом. Цей фактор стосується типів запитань, які можна ставити системі. Закриті системи відповідають на специфічні питання в таких галузях, як музика, погода або прогнозування, де система пристосована для відповідей на питання в обмеженому обсязі. Історично склалося так, що на початковому етапі основна увага була зосереджена на системах із закритими доменами, таких як LUNAR і BASEBALL.

Однак сучасні дослідження зосереджуються переважно на системах контролю якості з відкритим доменом, щоб охопити ширший спектр типів запитань. Цей зсув можна пояснити двома основними факторами: технологічним прогресом і практичністю. Завдяки експоненціальному зростанню обчислювальних потужностей дослідники тепер можуть розробляти складніші системи, здатні розв'язувати проблеми, пов'язані з QA у відкритому домені.

1.3 Парадигми системвідповідей на запитання

Існує чотири основні парадигми сучасних відповідей на запитання [11]:

а) Фактоїдні відповіді на запитання (IR-based Factoid Question Answering)

Основна мета цієї парадигми - відповісти на запитання користувача, витягуючи відповідні короткі текстові сегменти з Інтернету або інших колекцій документів. На етапі обробки запитання з нього витягуються різні фрагменти інформації. Тип відповіді визначає категорію відповіді (особа, місце, час тощо), а запит визначає ключові слова, які використовуються інформаційно-пошуковою системою (IR) для пошуку відповідних документів.

б) Контроль якості обробки природної мови: Використання лінгвістичної інтуїції та методів машинного навчання для вилучення відповідей з отриманого фрагмента.

в) Відповіді на запитання на основі знань (Knowledge-based Question Answering):

Ця парадигма передбачає відповіді на запитання природною мовою шляхом їх відображення у запитах до структурованих баз даних. Логічна форма питання, як правило, представлена у вигляді запиту або може бути легко перетворена в нього. База даних може бути повною реляційною базою даних або простішими структурованими базами даних, такими як набори RDF-трійок. Семантичні аналізатори використовуються для відображення текстових рядків у логічні форми, при цьому найчастіше використовуються мови обчислення предикатів або запитів, такі як SQL або SPARQL.

г) Використання декількох джерел інформації:

Система Watson від IBM[3], яка перемогла в конкурсі Jeopardy, покладаючись на різноманітні ресурси для відповіді на запитання, є прикладом цієї парадигми. Етап обробки запитань включає синтаксичний аналіз, тегування

іменованих сутностей і вилучення зв'язків. Подібно до текстових систем, DeepQA (базовий фреймворк Watson) визначає фокус, тип відповіді (лексичний тип відповіді або LAT), виконує класифікацію питання і розділяє питання на частини. Потім виділяється фокус запитання, після чого відбувається класифікація за типом (наприклад, запитання на визначення, запитання з множинним вибором, головоломка або заповнення пропусків).

Далі, на етапі генерації варіантів відповідей, оброблене запитання поєднується із зовнішніми документами та джерелами знань, щоб створити кілька варіантів відповідей. Ці варіанти можуть бути отримані з текстових документів або структурованих баз знань. Згодом відповіді-кандидати піддаються оцінюванню з використанням різних джерел доказів. Лексичний тип відповіді відіграє вирішальну роль у цьому процесі оцінювання. Нарешті, на етапі об'єднання та оцінювання відповідей еквівалентні відповіді кандидатів об'єднуються, і процес ранжування виконується ітеративно, враховуючи ранжування класифікатора, вибираючи відповідний варіант назви як об'єднану відповідь, і повторно ранжуючи об'єднані відповіді.

1.4 Виклики під час надання відповідей на запитання

Системи, що відповідають на запитання, зіштовхуються з кількома проблемами:

1. Лексична прогалина: природна мова дає змогу виражати одне й те саме значення різними способами. Заповнення лексичного розриву має вирішальне значення, оскільки на питання можна відповісти лише тоді, коли визначені всі релевантні поняття. Розв'язавши цю проблему, система може значно збільшити кількість запитань, на які вона може успішно відповісти.

2. Неоднозначність: Неоднозначність виникає, коли фраза або слово має кілька значень. Це може відбуватися структурно і синтаксично, як, наприклад, у

фразі "літаки", або лексично і семантично, як у слові "банк". Неоднозначність може проявлятися як омонімія, коли один і той самий рядок позначає не пов'язані між собою поняття (наприклад кран під'йомний vs. кран з водою), або полісемія, коли один і той самий рядок позначає різні, але пов'язані між собою поняття (наприклад, банк як фінансова установа vs. банк як фізична будівля).

3. Багатомовність: Знання в Інтернеті виражаються різними мовами. У той час як ресурси RDF можуть бути описані кількома мовами за допомогою мовних тегів, вебдокументи не дотримуються єдиної мови. Крім того, користувачі можуть мати різні рідні мови. Очікується, що система відповідей на запитання розпізнає мову, яка використовується, і надає результати відповідно до неї, враховуючи багатомовні можливості.

2 ВИКОРИСТАННЯ НЕЙРОННИХ МОДЕЛЕЙ ДЛЯ ПОБУДОВИ СИСТЕМИ ЗАПИТАНЬ ТА ВІДПОВІДЕЙ

2.1 Сучасні системи запитань та відповідей та AI

Сьогодні QA системи майже завжди застосовуються разом зі штучним інтелектом. Обробка природної мови (NLP), пошук інформації, представлення знань і машинне навчання використовуються, щоб обробляти та розуміти питання і надавати точні та релевантні відповіді. За останні роки ці системи зазнали значного прогресу завдяки швидкому розвитку технологій штучного інтелекту.

Моделі запитань відповідей (Question-Answering Models) – це набір моделей як і машинного, так і глибинного навчання, що здатні виконувати це завдання за допомогою штучного інтелекту. Ці моделі можуть використовувати різні методи, такі як вилучення відповідей, генеративне перефразування або вибір варіантів, залежно від набору даних, на якому вони навчалися або конкретної проблеми, для якої вони були розроблені. Архітектура нейронної мережі також відіграє певну роль у їхній функціональності.

Побудова таких моделей вимагає глибокого розуміння структури мови, семантичного розуміння контексту і запитань, а також здатності точно знаходити фрази-відповіді, серед інших труднощів. Отже, навчання моделей для виконання цих завдань, безсумнівно, є складним завданням. На щастя, концепція уваги в нейронних мережах значно допомогла у розв'язанні цих проблем. Механізми уваги, подібні до тих, що містяться в RNN-мережах, таких як R-NET і FusionNet, продемонстрували значні покращення в задачах відповідей на запитання [12] [13]

2.2 Архітектура Transformer

Ключову роль у розв'язанні проблеми відіграла архітектура Transformer, яка використовує механізм само уваги (self-attention). У 2017 Google випустила наукову статтю “Attention is all you need”[1], яка стала великим кроком вперед у використанні механізму уваги, бувши основним удосконаленням моделі під назвою "Transformer". Найвідоміші сучасні моделі, що з'являються в завданнях NLP, складаються з десятків трансформерів або деяких їхніх варіантів, наприклад, GPT-2 або BERT.

Трансформер – це архітектура моделі, що відмовляється від рекурентності та повністю покладається на механізм уваги для побудови глобальних залежностей між входом та виходом. Це перша трансдукційна модель, яка повністю покладається на само уважність для обчислення представлень своїх входів і виходів без використання вирівняних за послідовністю RNN або згортки

2.2.1 Переваги трансформерів

У задачах перекладу від послідовності до послідовності, таких як нейронний машинний переклад, початкові пропозиції базувалися на використанні RNN в архітектурі кодер-декодер. Ці архітектури мають велике обмеження при роботі з довгими послідовностями, їх здатність зберігати інформацію з перших елементів втрачалася, коли в послідовність включалися нові елементи. У кодері прихований стан на кожному кроці асоціюється з певним словом у вхідному реченні, зазвичай одним з останніх. Тому, якщо декодер отримає доступ лише до останнього прихованого стану кодера, він втратить релевантну інформацію про перші елементи послідовності.

Тоді для подолання цього обмеження було введено нову концепцію - механізм уваги. Замість того, щоб звертати увагу на останній стан кодера, як це зазвичай робиться з RNN, на кожному кроці декодера ми розглядаємо всі стани кодера, маючи доступ до інформації про всі елементи вхідної послідовності. Це те, що робить механізм уваги, він витягує інформацію з усієї послідовності,

зважену суму всіх минулих станів кодера. Це дає змогу декодера надавати більшу вагу або важливість певному елементу вхідних даних для кожного елемента вихідних даних. Навчання на кожному кроці фокусується на правильному елементі вхідних даних, щоб передбачити наступний елемент вихідних даних. Але цей підхід продовжує мати важливе обмеження, кожна послідовність повинна оброблятися по одному елементу за раз. І кодер, і декодер повинні чекати завершення n-1 кроків, щоб обробити n-й крок. Тому при роботі з великими масивами це дуже трудомісткий і неефективний в обчислювальному плані метод.

Модель Transformer витягує ознаки для кожного слова, використовуючи механізм само уваги, щоб з'ясувати, наскільки важливими є всі інші слова в реченні стосовновищезгаданого слова. Для отримання цих ознак не використовуються жодні рекурентні одиниці, а лише зважені суми та активації, тому вони можуть бути дуже розпаралеленими та ефективними.

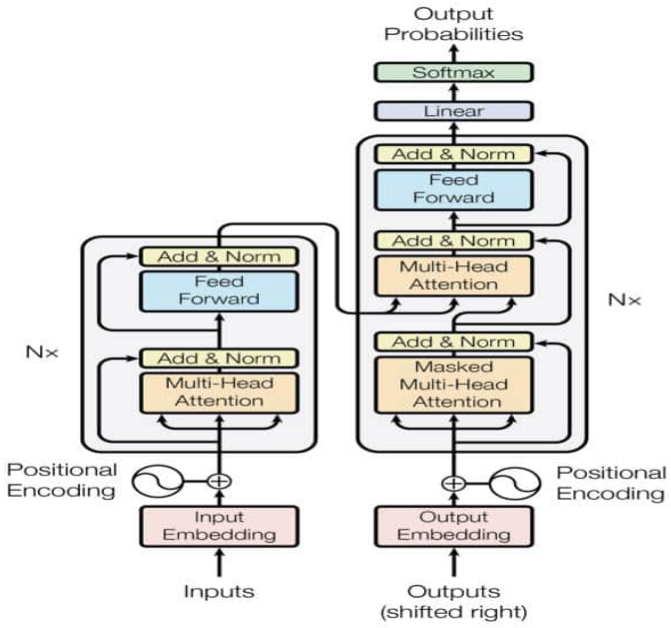

Рисунок 2.1.1 – Архітектура моделі трансформера

Ми бачимо, що ліворуч зображено модель кодера, а праворуч - декодера. Обидві містять основний блок "мережі уваги та зворотного зв'язку", що

повторюється N разів. Але спочатку сфокусуємось на основній концепції: механізм само уваги.

2.3 Механізм уваги

Само уважність це операція послідовності: послідовність векторів надходить на вхід і послідовність векторів виходить на вихід. Назвемо вхідні вектори x1, x2,..., xt і відповідні вихідні вектори y1, y2,..., yt. Всі вектори мають розмірність k. Щоб отримати вихідний вектор yi, операція само уваги просто бере середньозважене значення всіх вхідних векторів, найпростішим варіантом є точковий добуток. [4].

У механізм само уваги нашої моделі нам потрібно ввести три елементи: Запити, Значення та Ключі. Кожен вхідний вектор використовується в механізмі само уваги трьома різними способами: запит, ключ і значення. У кожній ролі він порівнюється з іншими векторами, щоб отримати власний вихід yi (Запит), отримати j-й вихід yj (Ключ) і обчислити кожен вихідний вектор після встановлення ваг (Значення).

Щоб отримати ці ролі, нам потрібні три вагові матриці розмірністю k x k і обчислити три лінійні перетворення для кожного xi:

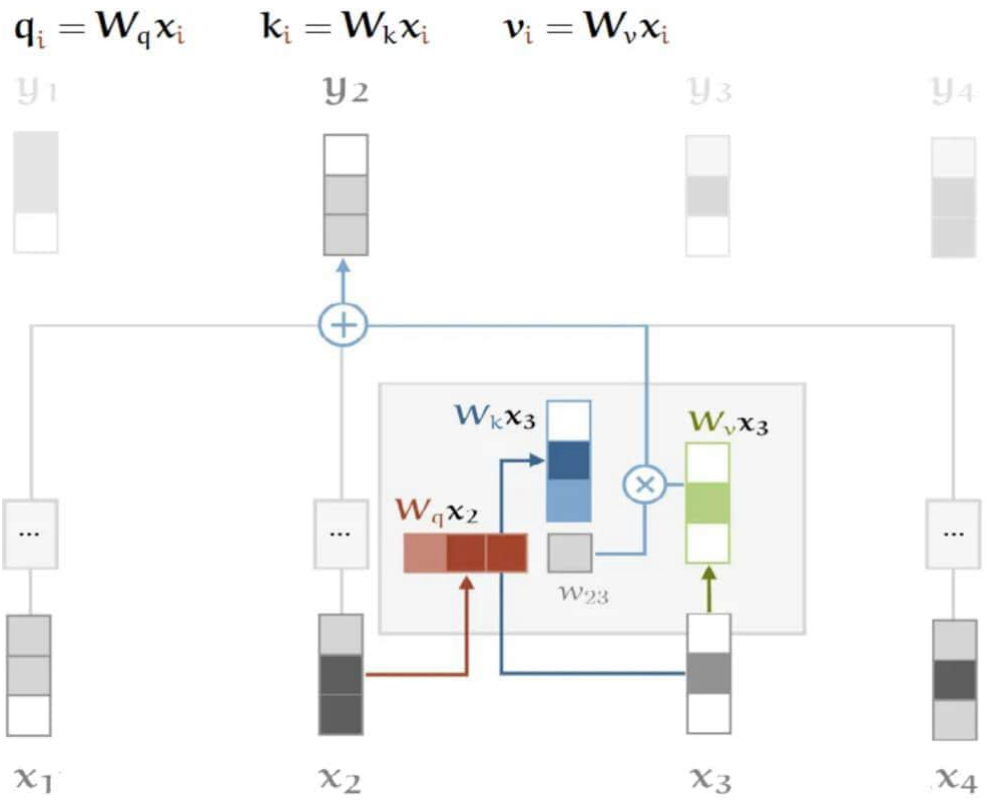

Рисунок 2.2.1 Робота механізму уваги з векторами q, k, v [4]

Ці три матриці зазвичай відомі як K, Q і V, три вагові шари, що навчаються, які застосовуються до одного і того ж закодованого входу. Оскільки кожна з цих трьох матриць надходить з одного і того ж входу, ми можемо застосувати механізм уваги вхідного вектора до самого себе, "само уваги".

Вхідні дані складаються із запитів і ключів розмірності dk та значень розмірності dv. Ми обчислюємо точковий добуток запиту з усіма ключами, ділимо кожен на квадратний корінь з dk і застосовуємо функцію softmax для отримання ваг значень [1].

Потім ми використовуємо матриці Q, K і V для обчислення балів за увагу. Оцінки вимірюють, скільки уваги потрібно зосередити на інших місцях або словах вхідної послідовності відносно слова в певній позиції. Тобто, ми оцінюємо точковий добуток вектора запиту на ключовий вектор відповідного

слова. Так, для позиції 1 обчислюємо точковий добуток (.) q1 і k1, потім q1. k2, q1. k3 і так далі.

Далі ми застосовуємо "масштабований" коефіцієнт, щоб мати більш стабільні градієнти. Функція softmax не може працювати належним чином при великих значеннях, що призводить до зникнення градієнтів і уповільнення навчання. Після операції "softmax" ми множимо на матрицю значень, щоб зберегти значення слів, на яких ми хочемо зосередитися, і мінімізуємо або видаляємо значення для нерелевантних слів (їх значення в матриці V повинно бути дуже малим).

На рисунку представлена форма для обчислення векторів уваги:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Рисунок 2.2.2 Формула для обрахунку механізму уваги [1]

2.4 Multi-head Attention

Щоб надати само уважності більшої дискримінаційної здатності, потрібно об'єднати кілька голів само уваги та розділити вектори слів на фіксовану кількість (h, кількість головок) фрагментів. Потім само уважність застосовується до відповідних фрагментів, використовуючи Q, K і V підматриці. Це дає h різних вихідних матриць оцінок. [4]

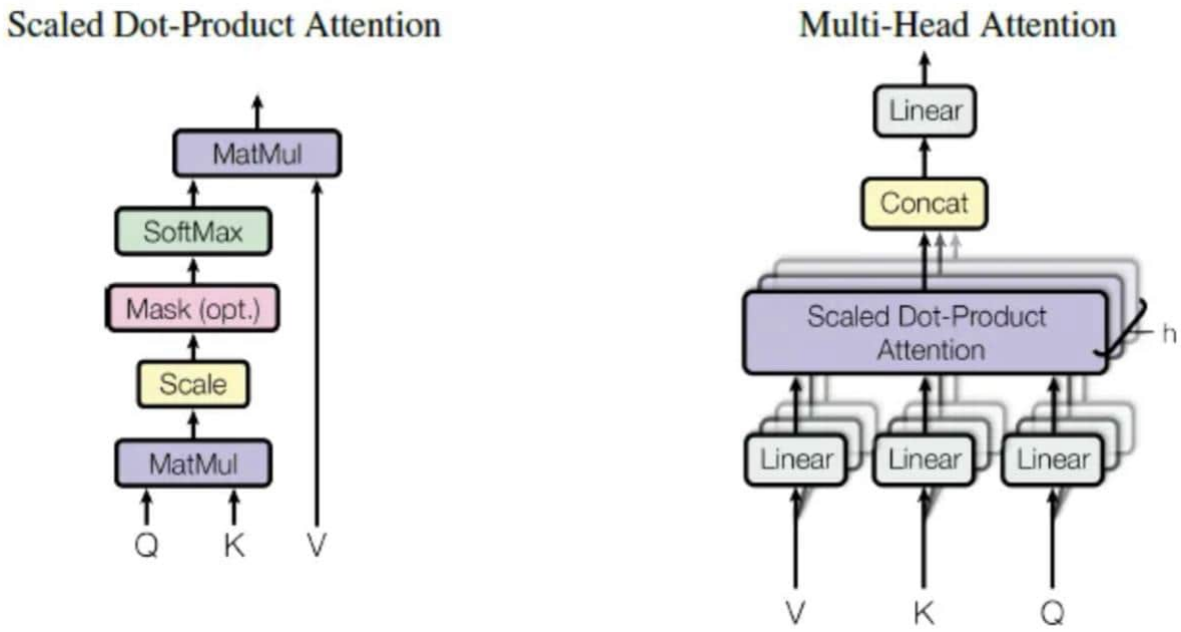

Рисунок 2.3.1 Механізм уваги для мультиголів (Multi-Head attention) [1]

Але наступний шар (шар прямого поширення) очікує лише одну матрицю, вектор для кожного слова, тому після обчислення точкового добутку кожної голови уваги ми об'єднуємо вихідні матриці та множимо їх на додаткову вагову матрицю Ця підсумкова матриця фіксує інформацію від усіх головок уваги.

2.5 Позиційне кодування (Positional Encoding)

Порядок слів у реченні - це проблема, яку потрібно вирішити в цій моделі, тому що мережа і механізм само уваги є інваріантними до перестановок. Якщо перемішати слова у вхідному реченні, то отримаємо ті ж самі рішення.

Задачею позиційного кодування є створення представлення позиції слова у реченні, і додавання його до вбудованих слів. Позиційні кодування мають ідентичні розмірності зі вставками, а тому можуть їх підсумовувати.

Отже, в моделі застосовується функція для відображення позицій у реченні у вектор з реальним значенням. Мережа навчиться використовувати цю інформацію.

Модель для обрахунків використовує синусоїдальну функцію:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Рисунок 2.4.1 Синусоїдальна функція [1]

2.6 BERT (Bidirectional Encoder Representations from Transformers)

Архітектура Трансформера складається з двох частин – Декодера та Кодера. Саме частина з кодером є основним компонентом моделі BERT, що використовується в даній роботі.

Стаття BERT (Bidirectional Encoder Representations from Transformers) [5] опублікована дослідниками з Google AI Language, викликала ажіотаж у спільноті машинного навчання, представивши найсучасніші результати в широкому спектрі завдань NLP, включаючи Answering Question Answering (SQuAD v1.1), Natural Language Inference (MNLI) та інші.

Ключовою технічною інновацією BERT є застосування двонапрямленого навчання Transformer, популярної моделі уваги, для моделювання мови. Це відрізняється від попередніх досліджень, які розглядали послідовність тексту зліва направо або комбінували навчання зліва направо та справа наліво. Результати статті показують, що двонапрямлена модель мови може мати глибше відчуття мовного контексту, ніж однонапрямний мовні моделі. У статті “Attention is all you need” [1] дослідники детально описують нову методику під назвою Masked LM (MLM), яка дає змогу двонапрямлене навчання в моделях, в яких раніше це було неможливо.

2.6.1 Принцип роботи BERT

BERT використовує Transformer - механізм уваги, який вивчає контекстуальні зв'язки між словами в тексті. У своїй звичайній формі Transformer складається з двох окремих механізмів - кодера, який зчитує введений текст, і декодера, який виробляє передбачення для завдання. Оскільки метою BERT є створення мовної моделі, необхідним є лише механізм кодування.

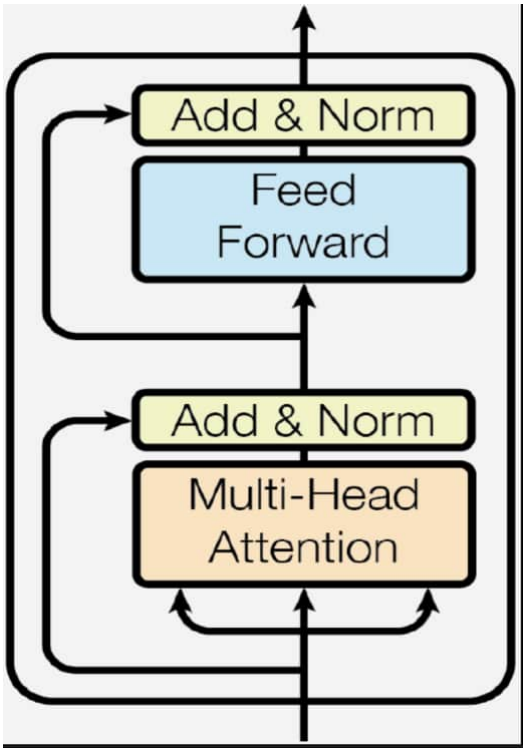

Рисунок 2.6.1.1 Зображення кодера зі статті “Attention is all you need”

На відміну від однонапрямлених моделей, які зчитують введений текст послідовно (зліва направо або справа наліво), кодер Transformer зчитує всю послідовність слів одразу. Тому він вважається двонапрямленим, хоча точніше було б сказати, що він ненаправлений. Ця характеристика дає змогу моделі вивчати контекст слова на основі всього його оточення (ліворуч і праворуч від слова).

Під час навчання мовних моделей виникає проблема визначення мети передбачення. Багато моделей передбачають наступне слово в послідовності (наприклад, "Дитина прийшла додому з ___"), що є цілеспрямованим підходом,

який по суті обмежує контекстне навчання. Щоб подолати цю проблему, BERT використовує дві стратегії навчання:

1. Моделювання замаскованої мови (Masked LM)
2. Передбачення наступних речень (NSP)

2.6.2 Моделювання замаскованої мови (Masked LM)

Перед тим, як завантажити послідовності слів у BERT, 15% слів у кожній послідовності замінюються лексемою [MASK]. Потім модель намагається передбачити початкове значення замаскованих слів на основі контексту, наданого іншими, незамаскованими словами в послідовності. Говорячи технічною мовою, передбачення вихідних слів вимагає:

1. Додавання шару класифікації над виходом кодера.
2. Множення вихідних векторів на матрицю вбудовування, перетворення їх у розмірність словника.
3. Обчислення ймовірності кожного слова в словнику за допомогою softmax.

Функція втрат BERT враховує лише передбачення замаскованих та ігнорує передбачення незамаскованих слів. Як наслідок, модель сходиться повільніше, ніж спрямовані моделі, але ця характеристика компенсується кращим розумінням контексту.

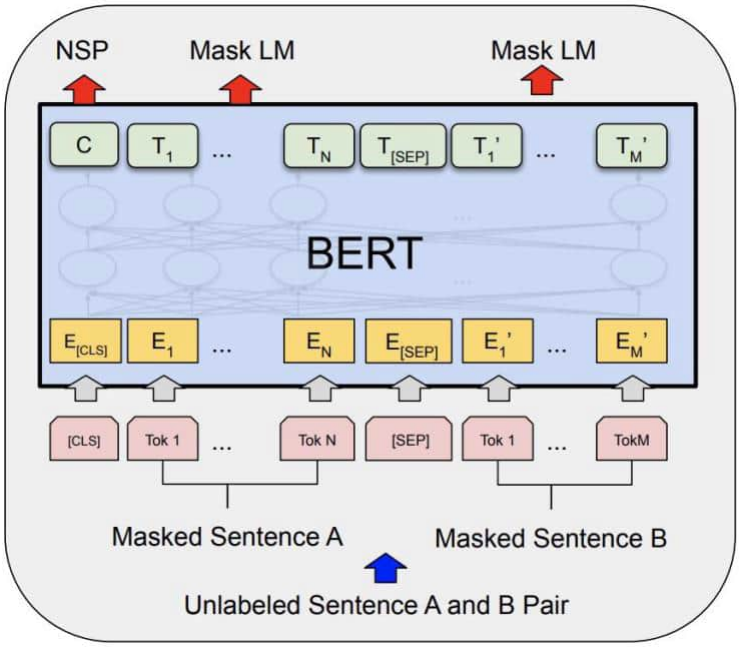

Рисунок 2.6.2.1 Місце Masked LM в архітектурі BERT

2.6.3 Передбачення наступного речення (NSP)

У процесі навчання BERT модель отримує на вхід пари речень і вчиться передбачати, чи є друге речення в парі наступним реченням у вихідному документі. Під час навчання 50% вхідних даних - це пара, в якій друге речення є наступним реченням у вихідному документі, тоді як в інших 50% випадкове речення з корпусу вибирається як друге речення. Припускається, що випадкове речення буде відокремлене від першого речення.

Щоб допомогти моделі розрізнити ці два речення під час навчання, вхідні дані обробляються наступним чином перед подачею на модель:

1. На початку першого речення вставляється лексема [CLS], а в кінці кожного речення - лексема [SEP].
2. До кожного речення додається вставка речення, яка вказує на речення A або речення B. Вставки речень за своєю концепцією подібні до вставок лексем зі словниковим запасом.
3. Позиційне вбудовування додається до кожної лексеми, щоб вказати її позицію у послідовності[1].

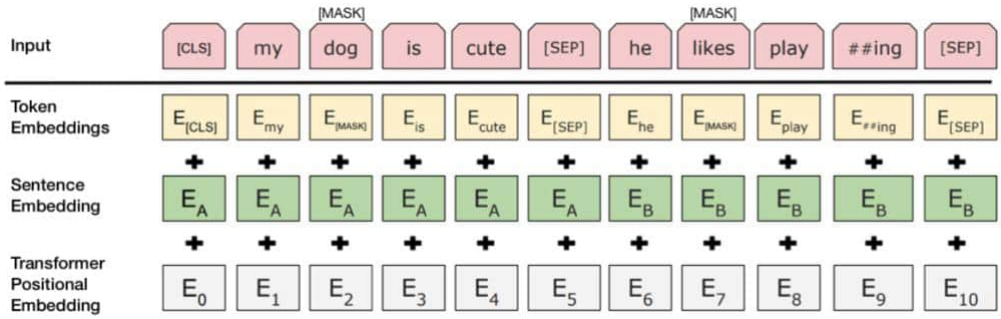

Рисунок 2.6.3.1 Приклад вхідних даних для BERT[5]

Щоб передбачити, чи дійсно друге речення пов'язане з першим, виконуються наступні кроки:

1. Вся вхідна послідовність проходить через модель Transformer.
2. Вихід токена [CLS] перетворюється на вектор форми 2×1, використовуючи простий шар класифікації (вивчені матриці ваг та зсувів).
3. Обчислення ймовірності IsNextSequence за допомогою softmax.

При навчанні BERT-моделі, Masked LM і Next Sentence Prediction навчаються разом, з метою мінімізації комбінованої функції втрат двох стратегій.

2.6.4 Донавчання BERT (Fine-tuning)

BERT від Google став зміною парадигми в моделюванні природної мови, зокрема, завдяки впровадженню парадигми попереднього навчання / точного налаштування (pre-training/fine-tuning): після попереднього навчання на великій кількості текстових даних без нагляду модель можна швидко налаштувати на конкретне наступне завдання з відносно невеликою кількістю міток, оскільки загальні лінгвістичні закономірності вже були вивчені під час попереднього навчання.

Автори Джейкоб Девлін (Jacob Devlin) та ін. пишуть, що точне налаштування BERT є "простим", просто додавши один додатковий шар після останнього шару BERT і навчивши всю мережу всього за кілька епох. Автори демонструють високу продуктивність на стандартних тестових завданнях NLP

GLUE, SQuAD і SWAG, які досліджують різні аспекти виведення природної мови, після тонкого налаштування всього за 2-3 епохи за допомогою оптимізатора ADAM зі швидкістю навчання від 1e-5 до 5e-5. Завдяки своєму неабиякому успіху ця парадигма попереднього навчання стала стандартною практикою в цій галузі.[6]

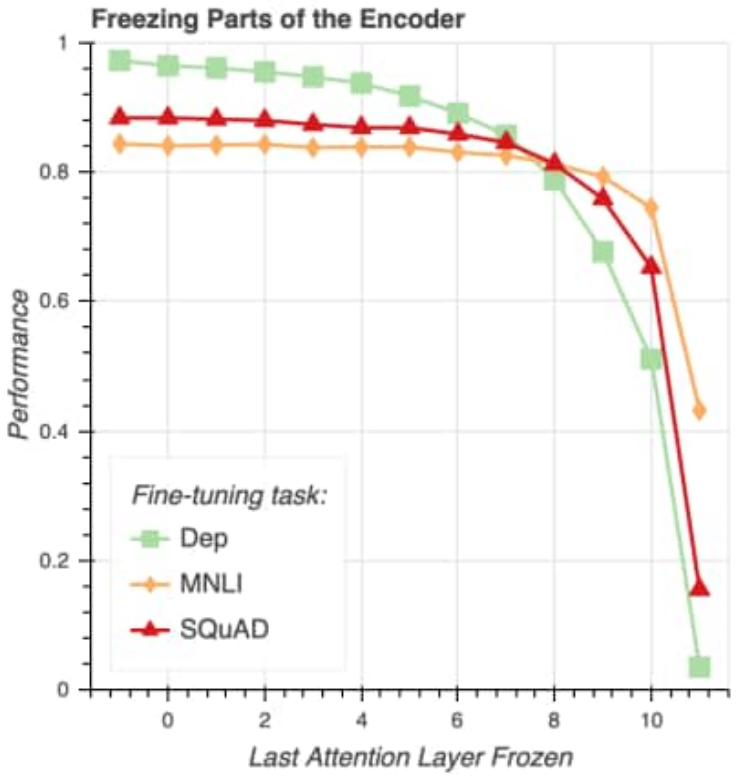

Рисунок 2.6.4.1 Приклад заморозки шарів BERT для різних наборів даних [6]

Як ми можемо бачити з рисунка 2.6.4.1, заморожування перших 8 шарів BERT мало впливає на ефективність точного налаштування на MNLI та SQUAD.

Принцип BERT полягає в тому, що ранні шари вивчають загальні лінгвістичні патерни, які мало відносяться до наступної задачі, тоді як пізні шари вивчають специфічні патерни для конкретної задачі. Цей принцип відповідає моделям глибокого комп'ютерного зору, де ранні шари вивчають загальні ознаки, такі як краї та кути, а пізні шари вивчають специфічні ознаки, такі як очі та ніс у випадку розпізнавання облич.

Цю гіпотезу експериментально підтвердила інша команда Google, Аміл Мерчант (Amil Merchant) та ін., у своїй роботі "What Happens To BERT Embeddings During Fine-tuning?"[6]. Одна з їхніх методик називається часткове заморожування: вони тримають ранні шари BERT замороженими, тобто фіксованими, під час процесу тонкого налаштування, і вимірюють, наскільки змінюється продуктивність на наступному завданні при зміні кількості заморожених шарів. Вони показують, що продуктивність як MNLI, так і SQuAD не помітно падає навіть при заморожуванні перших 8 з 12 шарів BERT (тобто при налаштуванні лише останніх 4).

Цей висновок підтверджує припущення про те, що останні шари є найбільш специфічними для задачі, а тому змінюються найбільше під час процесу точного налаштування, тоді як ранні шари залишаються відносно стабільними. Результати також означають, що фахівці можуть потенційно заощадити обчислювальні ресурси, заморозивши ранні шари замість того, щоб навчати всю мережу під час точного налаштування.

2.7 Види моделей BERT

Після представлення моделі BERT, відразу почали з’являтись її модифікації, які покращували той чи інший аспект моделі. Останнім часом було представлено кілька методів, які покращують метрики прогнозування або швидкість обчислень BERT, але не обидва ці показники.

Загальна таблиця за моделями:

	BERT	RoBERTa	DistilBERT	XLNet
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base:** 66	**Base**: ~110 **Large**: ~340
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large:** 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.	**Large:** 512 TPU Chips x 2.5 days; 5 times more than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT	2-15% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.	**Base**: 16 GB BERT data **Large**: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation	Bidirectional Transformer with Permutation based modeling

Таблиця 2.7.1 Результати порівняння різних архітектур

2.7.1 XLNet

XLNet[14] - це великий двонапрямлений трансформер, який використовує покращену методологію навчання, більший обсяг даних і більшу обчислювальну потужність, щоб досягти кращих, ніж у BERT, показників прогнозування на 20 мовних завданнях. [14]

Щоб покращити навчання, XLNet впроваджує перестановочне мовне моделювання, де передбачаються всі лексеми, але у випадковому порядку. Це відрізняється від маскованої мовної моделі BERT, де передбачаються лише замасковані (15%) лексеми. Також відрізняється від традиційних мовних моделей, де всі лексеми передбачаються в послідовному порядку, а не у випадковому. Перестановочне мовне моделювання допомагає моделі вивчати двонапрямлені зв'язки, а отже, краще обробляти залежності та відношення між словами. Крім того, як базову архітектуру було використано Transformer XL, який показав хорошу продуктивність навіть за відсутності навчання на основі перестановок.

XLNet навчався на більш ніж 130 ГБ текстових даних і 512 TPU-чіпах протягом 2,5 днів, що значно більше, ніж у BERT.

2.7.2 RoBERTa

Представлений у Facebook, надійно оптимізований підхід BERT RoBERTa - це перенавчання BERT з покращеною методологією навчання, на 1000% більшим обсягом даних та обчислювальною потужністю. [7]

Щоб покращити процедуру навчання, RoBERTa видаляє завдання передбачення наступного речення (NSP) з попереднього навчання BERT і вводить динамічне маскування, щоб замаскований токен змінювався під час навчальних епох. Також було виявлено, що більші розміри пакетного навчання є більш корисними для процедури навчання.

Важливо, що RoBERTa використовує 160 ГБ тексту для попереднього навчання, в тому числі 16 ГБ з корпусу книг та англійської Вікіпедії, що використовуються в BERT. Додаткові дані включають набір даних CommonCrawl News (63 мільйони статей, 76 ГБ), корпус вебтексту (38 ГБ) і Stories from Common Crawl (31 ГБ). Це в поєднанні з 1024 графічними процесорами V100 Tesla, що працювали протягом дня, призвело до попереднього навчання RoBERTa.

В результаті RoBERTa перевершує як BERT, так і XLNet в результатах бенчмарку GLUE:

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-
Ensembles on test (from leaderboard as of July 25, 2019)										
ALICE	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	**96.8**	**93.0**	67.8	91.6	**90.4**	88.4
RoBERTa	**90.8/90.2**	**98.9**	90.2	**88.2**	96.7	92.3	67.8	**92.2**	89.0	**88.5**

Рисунок 2.7.2.1 Результати моделей на різних бенчмарках [7]

З іншого боку, щоб зменшити час обчислень (навчання, прогнозування) BERT або споріднених моделей, природним вибором є використання меншої мережі для апроксимації продуктивності. Існує багато підходів, які можуть бути використані для цього, включаючи обрізання, дистиляцію та квантування, проте всі вони призводять до зниження метрик прогнозування.

2.7.3 DistilBERT

DistilBERT - це дистильована (наближена) версія BERT, що зберігає 97% продуктивності, але використовує лише половину параметрів (паперових). Зокрема, модель не має вбудовувань типу токенів, зберігає лише половину шарів з BERT від Google. DistilBERT використовує техніку, яка називається дистиляцією, що наближає BERT від Google, тобто велику нейронну мережу до меншої. Ідея полягає в тому, що після навчання великої нейронної мережі її повні вихідні розподіли можуть бути апроксимовані за допомогою меншої мережі. Це в деякому сенсі схоже на апостеріорну апроксимацію. Однією з ключових оптимізаційних функцій, що використовується для апостеріорної апроксимації в баєсівській статистиці, є дивергенція Кульбака-Лейбера, і, природно, вона також використовується і тут. [16]

2.7.4 XLM-RoBERTa

XLM-RoBERTa, що розшифровується як Cross-lingual Language Model - RoBERTa, є потужною мовною моделлю, яка привернула значну увагу в галузі обробки природної мови (NLP) і довела свою цінність для різних завдань NLP. Розроблена Facebook AI, XLM-RoBERTa базується на архітектурі моделі RoBERTa і розширює її можливості для ефективної обробки багатомовних даних.[15]

Однією з визначних особливостей XLM-RoBERTa є його здатність розуміти та генерувати текст декількома мовами, що робить його універсальним інструментом для міжмовних додатків. Вона була навчена на величезній

кількості багатомовних даних, що дозволяє їй фіксувати багаті мовні репрезентації та розуміти нюанси різних мов.

XLM-RoBERTa використовує можливості неконтрольованого попереднього навчання, де вона навчається на великих обсягах текстових даних без потреби в явних мітках. Завдяки цьому процесу попереднього навчання модель отримує глибоке розуміння мовних структур, контекстуальних залежностей і семантичних значень. Це розуміння дозволяє XLM-RoBERTa надзвичайно добре виконувати наступні завдання NLP, такі як класифікація текстів, розпізнавання іменованих об'єктів, аналіз настроїв і машинний переклад на різних мовах.

2.11 Відмінності RoBERTa від BERT

RoBERTa не використовує NSP, і ця модель навчається з набагато більшими мініпакетами та швидкістю навчання. Крім того, RoBERTa використовує іншу схему попереднього навчання і замінює словник BPE на рівні символів на токенізатор BPE на рівні байтів (подібно до GPT-2). До того ж нам не потрібно визначати, який токен належить до якого сегменту, оскільки в ньому також відсутні ідентифікатори token_type_ids. За допомогою токена розділення tokenizer.sep_token ми можемо легко розділити сегменти. [7]

Також замість 16 ГБ даних, які спочатку використовувалися для навчання BERT, RoBERTa навчається на великому наборі даних, який охоплює понад 160 ГБ нестисненого тексту. Набір даних для RoBERTa містить (16 ГБ) англійської Вікіпедії та книжкового корпусу, які використовуються в BERT. Додаткові дані включають корпус вебтекстів (38 ГБ), набір даних CommonCrawl News (63 мільйони статей, 76 ГБ) та Stories from Common Crawl (31 ГБ). Цей набір даних, а також 1024 графічних процесорів V100 Tesla, що працювали протягом дня, було використано для попереднього навчання RoBERTa.

Щоб створити RoBERTa, команда Facebook спочатку перенесла BERT з фреймворку глибокого навчання TensorFlow від Google на свій фреймворк PyTorch. RoBERTa навчається на повних речення без втрати NSP, динамічному маскуванні, великих мініпакета та більшому BPE на рівні байтів.

Ключові відмінності між RoBERTa і BERT можна підсумувати наступним чином:

- RoBERTa є повторною реалізацією BERT з деякими змінами ключових гіперпараметрів і незначними налаштуваннями вбудовування. Він використовує байтовий BPE як токенізатор (подібно до GPT-2) та іншу схему попереднього навчання.

- RoBERTa також навчається на довших послідовностях, тобто кількість ітерацій збільшується зі 100K до 300K, а потім до 500K.

- RoBERTa використовує більший словник BPE на рівні байтів з 50 тис. одиниць токенів замість словника BPE на рівні символів розміром 30 тис., який використовується в BERT.

- У навчальній задачі "Маскована мовна модель" (MLM) RoBERTa використовує динамічне маскування для генерації шаблону маскування кожного разу, коли послідовність подається на модель.

- RoBERTa не використовує ідентифікатори token_type_ids, і нам не потрібно визначати, яка лексема належить до якого сегмента. Просто відокремлюємо сегменти за допомогою токена-розділювача tokenizer.sep_token

- Завдання передбачення наступного речення (NSP) вилучається з процедури навчання.

- У навчанні RoBERTa використовуються більші мініпакети та швидкість навчання.

2.8 Статичне та динамічне маскування

Завдання маскованого моделювання мови в попередньому навчанні BERT полягає в тому, щоб випадковим чином замаскувати кілька лексем з кожної послідовності, а потім передбачити ці лексеми. Однак в оригінальній реалізації BERT послідовності маскуються лише один раз під час попередньої обробки. Це означає, що той самий шаблон маскування використовується для тієї самої послідовності на всіх кроках навчання.

Щоб уникнути цього, при повторній реалізації BERT автори продублювали навчальні дані 10 разів, щоб кожна послідовність була замаскована 10 різними шаблонами. Навчання відбувалося протягом 40 епох, тобто кожна послідовність навчалася для тих самих шаблонів маскування 4 рази.

На додаток до цього було випробувано динамічне маскування, коли шаблон маскування генерується щоразу, коли послідовність подається на модель.

Masking	**SQuAD 2.0**	**MNLI-m**	**SST-2**
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Рисунок 2.8.1 Порівняння статичного та динамічного підходів до маскування [7]

Вищезгадані результати показують, що повторна реалізація зі статичним маскуванням дає майже такі ж результати, як і оригінальний підхід до маскування BERT. Динамічне маскування має дещо кращі результати, ніж статичне маскування. Тому в RoBERTa для попереднього навчання використовується підхід динамічного маскування.

2.9 Представлення вхідних даних і передбачення наступного речення

В оригінальній статті про BERT припускається, що завдання передбачення наступного речення (NSP) є важливим для отримання найкращих результатів від моделі. Нещодавні дослідження поставили під сумнів необхідність цього завдання у попередньому навчанні.

Отже, зараз ми розглянемо різні типи представлення вхідних даних, які можна використовувати з BERT, і як вони допоможуть усунути завдання NSP у попередньому навчанні: [1]

1. Сегмент-пара + NSP: Це представлення вхідних даних, яке використовується в реалізації BERT. Кожен вхідний файл містить пару сегментів (сегментів, а не речень) з оригінального документа або з іншого документа, вибраних випадковим чином з ймовірністю 0,5, а потім вони тренуються для розпізнавання тексту або для завдання Natural Language Inference (NLI). Загальна комбінована довжина повинна бути < 512 лексем (що є максимальною фіксованою довжиною послідовності для моделі BERT).

2. Речення-пара + NSP: Те саме, що й представлення сегмент-пара, тільки з парами речень. Однак очевидно, що загальна довжина послідовностей тут буде набагато меншою, ніж 512 токенів. Тому використовується більший розмір пакети, щоб кількість лексем, оброблених за один крок навчання, була схожою на кількість лексем у сегментно-парному

поданні.

3. Повні речення: Вхідні послідовності складаються з повних речень з одного або декількох документів. Якщо один документ закінчується, то беруться речення з наступного документа і відокремлюються за допомогою додаткової лексеми-розділювача, доки довжина послідовності не перевищить 512 токенів.

4. Речення документа: Це те саме, що й повні речення, тільки послідовність не перетинає межі документів, тобто коли документ закінчується, речення з наступних документів не додаються до послідовності. Тут, оскільки довжини документів варіюються, розмір пакетів також варіюється, щоб відповідати кількості токенів у повних реченнях.

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (K = 6)	–/81.0	85.6	93.4	66.7

Рисунок 2.9.1 Порівняння продуктивності для різних представлень вхідних даних зі статті

Щодо порівняння результатів, то, по-перше, представлення сегмент-пара, яке спочатку використовувалося в роботі Девліна та ін., краще виконує подальші завдання, ніж представлення окремого речення (речення-пара). Однак

представлення "документ-речення" перевершує оригінальну модель BERT (BASE). Видалення мети NSP збігається або дещо покращує виконання наступних завдань.

2.10 Великі розміри пакетів

Попередня робота показала, що моделі Transformer і BERT підходять для великих розмірів пакетів. Великі розміри пакетів роблять оптимізацію швидшою і можуть покращити продуктивність кінцевої задачі при правильному налаштуванні (у випадку з цими моделями).

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

Рисунок 2.10.1 Порівняння продуктивності з різними розмірами пакетів зі статті [1]

Зауважте, що зі збільшенням розміру пакетів кількість навчальних проходів коригується, тобто задана послідовність в кінцевому підсумку буде оптимізована однакову кількість разів. Наприклад, розмір пакетів 256 для 1 млн кроків еквівалентний навчанню з розміром пакетів 2 тис. для 125 тис. кроків і з розміром пакетів 8 тис. для 31 тис. кроків.

2.11 Токенізація

Для токенізації RoBERTa використовує схему кодування Byte-Pair Encoding (BPE) на рівні байт зі словником, що містить 50 тис. кодувань, на відміну від BERT's BPE на рівні символів зі словником у 30 тис. кодувань.

3. ДАНІ ДЛЯ ТРЕНУВАННЯ СИСТЕМИ ЗАПИТАНЬ ТА ВІДПОВІДЕЙ

3.1 Підбір набору даних

Набори даних відіграють фундаментальну роль у процесі машинного навчання, бо використовуються як вхідні дані, на основі яких моделі навчаються, перевіряються та тестуються. Основна мета використання наборів даних - зафіксувати та вивчити закономірності, взаємозв'язки та залежності в даних, що дає змогу моделі робити точні прогнози або класифікації на нових, невидимих прикладах даних.

На етапі навчання набір даних використовується для оптимізації параметрів моделі шляхом мінімізації помилок або втрат між прогнозами моделі та істинними мітками. Чим більш репрезентативним і різноманітним є набір даних, тим краще модель може навчатися та узагальнювати свої висновки. Крім того, набори даних мають вирішальне значення для оцінки продуктивності моделі на етапах валідації та тестування, оскільки вони є еталоном для оцінки точності, запам'ятовування та інших метрик продуктивності моделі.

Якість набору даних безпосередньо впливає на продуктивність і надійність моделей машинного навчання. Неякісний набір даних, який містить помилкові або суперечливі дані, може призвести до упереджених або ненадійних прогнозів. Проблеми з якістю даних можуть виникати через різні фактори, включно з помилками збору даних, пропущеними значеннями, викидами або дисбалансом класів.

З іншого боку, високоякісні набори даних сприяють створенню більш точних і надійних моделей. Добре оброблений набір даних гарантує, що навчальні дані відображають справжній розподіл проблемного простору та зменшують ризик надмірної або недостатньої підготовки. Крім того, високоякісні набори даних допомагають виявити основні закономірності та

взаємозв'язки, що дає змогу моделі узагальнювати та робити точні прогнози на основі нових даних.

Збір наборів даних для завдань машинного навчання може бути складним і важким процесом. Для забезпечення репрезентативності та надійності набору даних необхідно враховувати кілька факторів. Ці міркування містять наступні етапи:

- Збір даних: Визначення відповідних джерел і методів збору даних, які точно відображають проблемну область. Це може включати вебскрепінг, збір даних з датчиків, опитування або співпрацю з експертами в цій галузі.
- Попередня обробка даних: Очищення та підготовка даних для розв'язання таких проблем, як пропущені значення, викиди та невідповідності. На цьому етапі часто застосовують методи очищення даних, інженерію функцій і нормалізацію, щоб забезпечити придатність набору даних для аналізу.
- Конфіденційність та етика даних: Забезпечення дотримання правил конфіденційності та етичних норм при зборі та обробці конфіденційних або персональних даних. Це включає отримання інформованої згоди, анонімізацію даних та впровадження відповідних заходів безпеки.
- Упередженість і справедливість даних: Усвідомлення та пом'якшення упереджень, які можуть існувати в даних, таких як демографічні упередження або упередження вибірки, для запобігання дискримінаційним результатам та забезпечення справедливості в машинному аналізі.

3.2 SQuAD 2.0 (Stanford Question Answering Dataset) перекладений на українську

Стенфордський набір даних для відповіді на запитання 2.0 (SQuAD 2.0) став ключовим ресурсом для навчання моделей відповіді на запитання. SQuAD 2.0 не лише надає еталонний набір даних, але й вводить питання, на які немає відповіді, що змушує моделі не лише давати точні відповіді, але й визначати, коли на питання не можна відповісти, виходячи із заданого контексту. [17]

3.3 Важливість SQuAD 2.0

SQuAD 2.0 пропонує кілька переконливих переваг для навчання моделей відповідей на запитання. Перш за все, це всеосяжний і різноманітний набір даних, який охоплює широкий спектр тем і типів запитань. Ця різноманітність гарантує, що моделі отримують доступ до широкого спектра мовних шаблонів, контекстуальних нюансів і специфічних знань про предметну область. Тренуючись на такому багатому наборі даних, моделі, що відповідають на запитання, можуть розвинути глибоке розуміння різних мовних конструкцій, що призводить до покращення продуктивності та узагальнення на невидимих даних.

Однією з визначних особливостей SQuAD 2.0 є включення питань без відповіді. У реальних сценаріях не всі питання мають однозначну відповідь у певному контексті. Включаючи питання без відповіді в набір даних, SQuAD 2.0 імітує ці реальні складнощі, кидаючи виклик моделям для точного визначення та обробки таких випадків. Це доповнення заохочує моделі демонструвати кращі здібності до міркувань, більш всебічно аналізувати контекст і приймати обґрунтовані рішення щодо відповідальності. Таким чином, навчання роботі з SQuAD 2.0 дає змогу моделям, що відповідають на запитання, обробляти ширший спектр сценаріїв і підвищує їхню надійність у практичному застосуванні.

3.4 Методи попередньої обробки та архітектури моделей

Щоб використати весь потенціал SQuAD 2.0, необхідні відповідні методи попередньої обробки. Етапи очищення та нормалізації даних застосовуються для

забезпечення цілісності та узгодженості набору даних. Такі методи, як видалення дублікатів, обробка зашумлених або неповних даних і нормалізація тексту, сприяють підвищенню загальної якості набору даних. Крім того, методи функціональної інженерії можна використовувати для вилучення релевантної інформації та ефективного захоплення контексту, що дає змогу моделям робити точніші прогнози.

З точки зору архітектури моделей, моделі на основі трансформерів довели свою високу ефективність у задачах пошуку відповідей на запитання. Такі моделі, як BERT (Bidirectional Encoder Representations from Transformers), RoBERTa та ALBERT досягли чудових результатів, використовуючи силу механізмів само уваги та контекстних вбудовувань. Ці моделі фіксують складні взаємозв'язки між словами та реченнями, що дає змогу їм більш точно інтерпретувати та розуміти контекст. Навчені на SQuAD 2.0, трансформер моделі демонструють чудову продуктивність у відповідях на запитання та виявленні запитів, на які немає відповіді, що свідчить про ефективність цих архітектур у машинному зчитуванні.

3.5 Метрики оцінювання та аналіз продуктивності

Для оцінки продуктивності моделей відповідей на запитання, навчених на SQuAD 2.0, потрібні відповідні метрики. Традиційно для вимірювання точності передбачення відповідей використовуються такі показники, як точний збіг (EM) і оцінка F1. EM оцінює, наскільки точно відповідь моделі збігається з істинною відповіддю, тоді як показник F1 кількісно оцінює збіг між прогнозованою та істинною відповідями на основі співставлення на рівні токенів. Ці показники дають уявлення про здатність моделі давати точні та вичерпні відповіді.

Однак, через включення в SQuAD 2.0 питань без відповіді, для оцінки моделі потрібні додаткові метрики оцінки.

Головною перепоною використання оригінального SQuAD 2.0 було те, що він є доступний тільки англійською мовою. А оскільки, мета даної роботи натренувати модель з використанням українського набору даних, то необхідно було знайти або україномовний набір даних, або перекладений англомовний. Саме другий варіант я використав у цій роботі, взявши дані з роботи Context-Based Question-Answering System for the Ukrainian Language [8]. У цій роботі дослідники для тренування BERT моделі для завдання запитань та відповідей переклали оригінальний SQuAD 2.0, а також об’єднали його з набором даних Sberbank Data Science Journey 2017 dataset, який також було перекладено на українську.

4. ВИКОРИСТАННЯ ХМАРНИХ ОБЧИСЛЕНЬ ДЛЯ ТРЕНУВАННЯ МОДЕЛЕЙ

4.1 Amazon Web Services

AWS, також відома як Amazon Web Services, - це комплексна платформа хмарних обчислень, яка надається компанією Amazon. Вона охоплює поєднання пропозицій інфраструктури як послуги, платформи як послуги та пакетного програмного забезпечення як послуги. Ці сервіси надають організаціям різні інструменти, такі як обчислювальні потужності, сховища баз даних та послуги доставлення контенту. [18]

Amazon Web Services запустив свої перші вебсервіси у 2002 році на базі внутрішньої інфраструктури, яку Amazon побудував для обслуговування своїх роздрібних онлайн-операцій. AWS однією з перших запровадила модель оплати за фактом використання, що дає змогу користувачам масштабувати свої обчислювальні ресурси, сховище та пропускну здатність відповідно до їхніх потреб.

Функціональність AWS організована в різні сервіси, кожен з яких може бути налаштований по-різному, щоб задовольнити конкретні вимоги користувачів. Користувачі мають доступ до опцій конфігурації та індивідуальних карт серверів для кожного сервісу AWS, що дає змогу їм відповідно налаштувати свої параметри. AWS надає потужну та масштабовану інфраструктуру для навчання моделей у галузі машинного навчання та штучного інтелекту. Завдяки широкому спектру послуг, AWS пропонує різноманітні інструменти та ресурси для полегшення навчання моделей.

4.2 AWS Lambda

AWS Lambda - це без серверний, керований подіями обчислювальний сервіс, що дає змогу вам виконувати код практично для необмеженого типу

застосунку або сервісу без надання серверів та їх обслуговування. Цей сервіс легко запускати без управління інфраструктурою. Через те, що Lambda використовує без серверну архітектуру, тобто нам не потрібно турбуватися про забезпечення або керування серверами. AWS обробляє всю серверну інфраструктуру, включаючи масштабування, виправлення та моніторинг. Lambda автоматично масштабує додаток на основі обсягу вхідних запитів. Він надає необхідні ресурси для обробки робочого навантаження і зменшує масштабування, коли немає вхідного трафіку, забезпечуючи ефективне використання ресурсів. Додатки запускаються за мілісекунди після спрацювання події, а тисячі функцій можуть працювати паралельно.

За допомогою AWS Lambda можна розбити дані на менші фрагмент і кількома Lambda-функціями паралельно обробити кожен фрагмент задля великомасштабного або обчислювально інтенсивного навчання.

Функції AWS Lambda можна використовувати для попередньої обробки та перетворення даних перед подачею їх у навчальний конвеєр. Наприклад, ви можете написати Lambda-функції для очищення даних, інженерії функцій або доповнення даних. Lambda-функції можуть запускатися такими подіями, як завантаження файлів до Amazon S3 або повідомлення з черги Amazon Simple Queue Service (SQS), що забезпечує безперешкодну інтеграцію з джерелами даних.

Загалом, AWS Lambda спрощує процес створення масштабованих, керованих подіями додатків, абстрагуючись від керування сервером. Це дає змогу розробника зосереджуватися на написанні коду, підвищує гнучкість та зменшує операційні витрати. Незалежно від того, чи потрібно нам обробляти дані, запускати фонові завдання або створювати API, застосовуючи сервіс Lambda ми можемо швидко знайти гнучке та економічно ефективне рішення для без серверних обчислень в екосистемі AWS.

Після процесу навчання моделі Lambda-функції допомагають в оцінці продуктивності навченої моделі. Lambda-функції можна застосовувати для виконання таких завдань, як перехресна перевірка, обчислення метрик оцінки або створення звітів про перевірку. Ці функції можна запускати вручну або запланувати на певні інтервали часу за допомогою таких сервісів, як Amazon CloudWatch Events.

Щоб підготувати свої дані, їх необхідно попередньо очистити, трансформувати та впорядкувати. Після цього їх можна зберігати у сховищі Amazon Simple Storage Service (S3), яке забезпечує масштабоване та довговічне зберігання об'єктів.

4.3 Amazon S3

Amazon Simple Storage Service (Amazon S3) - це сервіс для зберігання та захисту будь-яких обсягів даних для різних випадків використання, таких як озера даних, вебсайт, мобільні додатки, резервне копіювання та відновлення, архівування, корпоративні додатки, пристрої Інтернету речей та аналітика великих даних. Amazon S3 надає функції керування, щоб, організувати та налаштувати доступ до даних відповідно до поставлених вимог в завдання.

Наприклад, можна використовувати використовувати S3 Версії для зберігання декількох версій об'єкта в одному кошику, що дає змогу відновлювати об'єкти, які були випадково видалені або перезаписані.

4.4 Amazon API Gateway

Також в навчанні моделей використовується Amazon API Gateway - повністю керований сервіс, який полегшує розробникам створення, публікацію, підтримку, моніторинг та захист API будь-якого масштабу. API діють як "вхідні двері" для додатків для доступу до даних, бізнес-логіки або функціональності з ваших внутрішніх служб. Gateway надає можливість створити кінцеву точку API,

щоб приймати дані для навчання моделі. Кінцева точка API може перевіряти і попередньо обробляти вхідні дані, перш ніж передавати їх в процес навчання моделі. API Gateway в інтеграції з AWS Lambda, дає змогу нам виконувати власний код у відповідь на запити API. Lambda-функції можна використовувати для перетворення даних, розробки функцій або доповнення вхідних даних перед тим, як подавати їх у конвеєр навчання моделі. Цей без серверний підхід забезпечує масштабовану та гнучку попередню обробку даних.

API Gateway виконує всі завдання, пов'язані з прийомом і обробкою до сотень тисяч одночасних викликів API, включаючи управління трафіком, підтримку CORS, авторизацію і контроль доступу, дроселювання, моніторинг і управління версіями API.

4.5 Додаткові AWS сервіси для тренування нейронних моделей

AWS надає сервіси та інструменти, які можна використовувати для навчання моделей у машинному навчанні та штучному інтелекті.

Для великомасштабного навчання або навчання з інтенсивними обчисленнями можна використовувати, такі сервіси як AWS Batch або Amazon Elastic Inference. AWS Batch дає змогу запускати розподілені навчальні завдання на декількох екземплярах, автоматично керуючи виділенням і масштабуванням ресурсів. Amazon Elastic Inference допомагає оптимізувати витрати на висновок шляхом прискорення виводу на графічних процесорах для екземплярів EC2.

Моніторинг та оптимізація: Контроль процесу навчання можна здійснювати за допомогою таких інструментів, як Amazon CloudWatch, щоб відстежувати такі показники, як втрати при навчанні, точність і використання ресурсів. Також для організації робочого процесу можна використовувати AWS Step Functions та AWS Parallel Cluster для управління високопродуктивними обчислювальними кластерами для розподіленого навчання.

Розгортання моделі: Після успішного навчання та оцінки моделі її можна розгорнути за допомогою сервісів AWS, таких як Amazon SageMaker. SageMaker надає керований хостинг і можливості виведення, що дає змогу легко розгортати ваші навчені моделі як вебсервіси або інтегрувати їх у додатки.

4.6 Amazon SageMaker

Amazon SageMaker - це повністю керований сервіс машинного навчання, який спрощує наскрізний процес побудови, навчання та розгортання моделей машинного навчання.[9] Він надає повний набір можливостей, включаючи попередньо налаштовані ноутбуки Jupyter для дослідження даних та експериментів, керовані навчальні середовища та вбудовані алгоритми для поширених завдань машинного навчання. Платформа автоматизує виснажливу роботу з побудови готового до виробництва конвеєра штучного інтелекту (ШІ).

AWS SageMaker використовує інтегровані технології для автоматизації трудомістких ручних процедур, зменшуючи при цьому кількість людських помилок та витрати на обладнання.

AWS SageMaker підвищує продуктивність проєкт машинного навчання, допомагає у створенні та управлінні обчислювальними екземплярами в найкоротші терміни, перевіряє вихідні дані перед автоматичним створенням, розгортанням і навчанням моделей з повною видимістю. Це скорочує витрати на розробку моделей машинного навчання на 70%.

Інструментарій AWS SageMaker містить компоненти ML-моделювання. У шаблонах SageMaker функції програмного забезпечення абстраговані. Вони забезпечують платформу для побудови, хостингу, навчання та розгортання моделей машинного навчання в масштабі в публічній хмарі Amazon.

Також, коли потреби в навчанні моделей зростають, AWS дає змогу легко масштабувати інфраструктуру. автоматизувати весь процес, використовуючи

сервіси та інструменти AWS, такі як AWS Step Functions, AWS Lambda або AWS Batch, щоб створити автоматизовані робочі процеси для введення даних, навчання моделей і розгортання.

5. ПРАКТИЧНА ЧАСТИНА

5.1Розробка та тренування моделі

Для розробки моделі, як і зазначено в теоретичній частині, було використано сервіс AWS, а саме Amazon Sagemaker [9]. З його допомогою можна використовувати хмарні обчислення для прискореного тренування моделі.

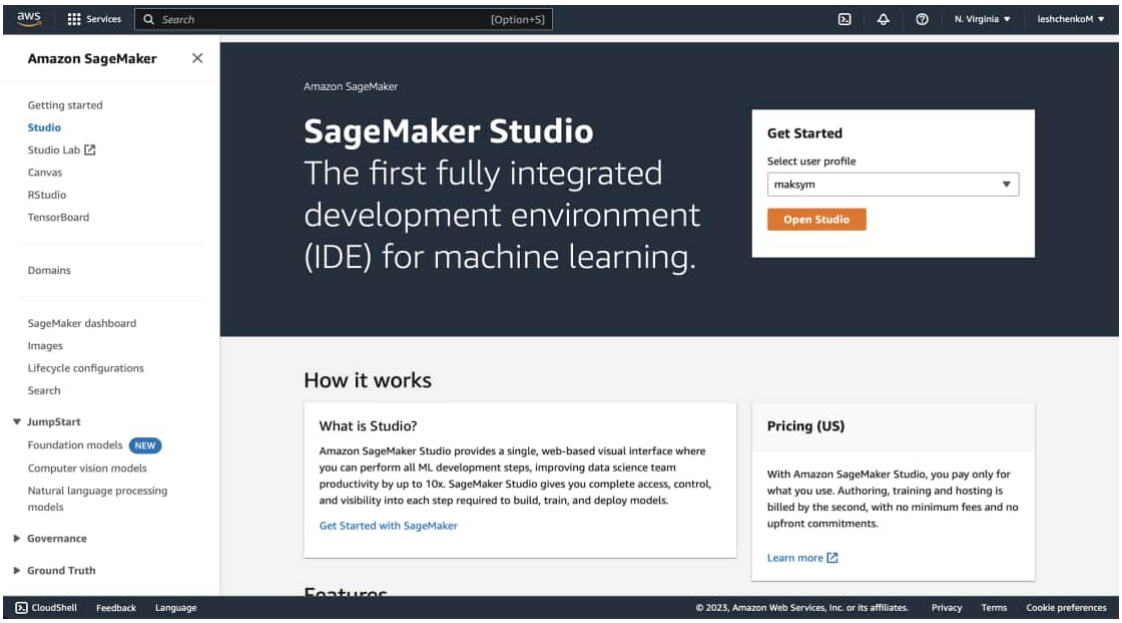

Рисунок 5.1.1 Інтерфейс Amazon Sagemaker

Інтерфейс середовища розробки є інтерфейсом Jupyter Studio, що дає змогу не витрачати час на вивчення системи з нуля, а відразу працювати в добре знайомому середовищі.

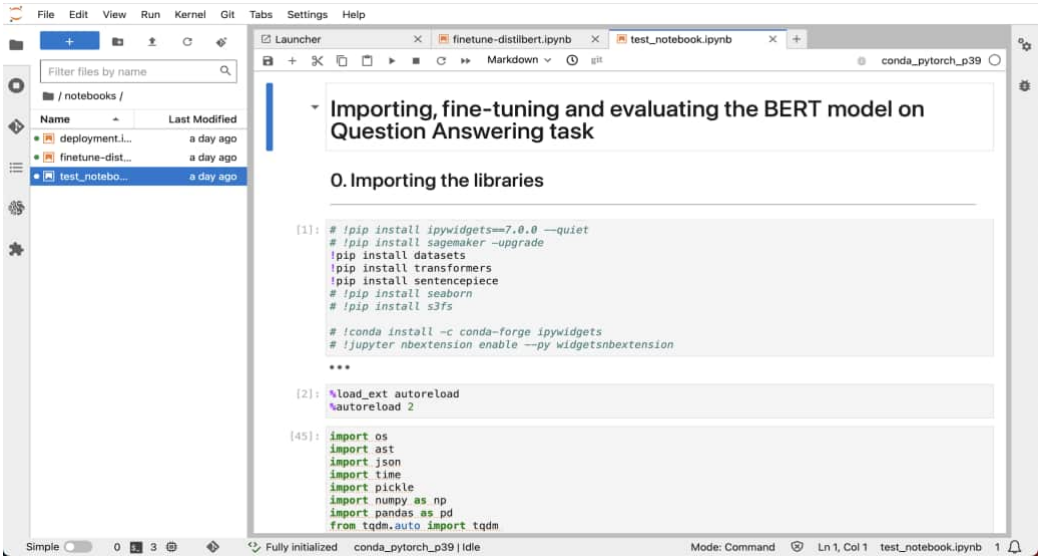

Рисунок 5.1.2 Інтерфейс середовища розробки Sagemaker Studio

Завантаживши всі необхідні для розробки бібліотеки, переходимо до завантаження та обробки даних. Дані попередньо було збережено у хмарному сховищі S3, яке надає Amazon.

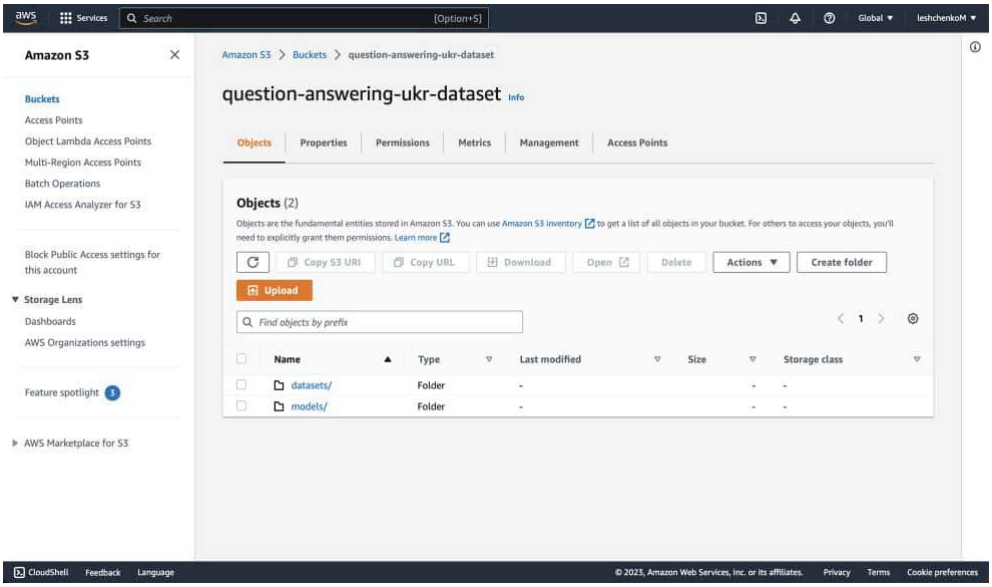

Рисунок 5.1.3 Створене хмарне сховище S3 та збережені в ньому дані

Після додавання даних у сховище, їх потрібно завантажити у себе в середовищі розробки:

0.1. Downloading the Squad2.0. dataset from the S3 AWS storage

```
[21]: content_object = s3.Object('question-answering-ukr-dataset', train_dataset+'/train-v2.0.json')
file_content = content_object.get()['Body'].read().decode('utf-8')
train_json_content = json.loads(file_content)

content_object = s3.Object('question-answering-ukr-dataset', test_dataset+'/dev-v2.0.json')
file_content = content_object.get()['Body'].read().decode('utf-8')
val_json_content = json.loads(file_content)
```

```
[22]: train = pd.DataFrame.from_dict(train_json_content)
test = pd.DataFrame.from_dict(val_json_content)
```

```
[23]: train.shape, test.shape
```

```
[23]: ((39221, 2), (35, 2))
```

Рисунок 5.1.4 Зчитування даних із S3 середовища

Після отримання даних було створено функцію для перетворення формату JSON в більш зручний для обробки формат бібліотеки pandas – DataFrame:

```
[24]: def create_df_from_json(df: pd.DataFrame, type: str = None) -> pd.DataFrame:
          contexts = []
          questions = []
          is_impossible = []
          answers_text = []
          answers_start = []

          for i in range(df.shape[0]):
              topic = df.iloc[i, 1]['paragraphs']
              for sub_para in topic:
                  for q_a in sub_para['qas']:
                      if q_a['answers']:
                          for answer in q_a['answers']:
                              is_impossible.append(q_a['is_impossible'])
                              contexts.append(sub_para['context'])
                              questions.append(q_a['question'])
                              answers_text.append(answer['text'])
                              answers_start.append(answer['answer_start'])
                      else:
                          is_impossible.append(q_a['is_impossible'])
                          contexts.append(sub_para['context'])
                          questions.append(q_a['question'])
                          answers_text.append([])
                          answers_start.append([])
              data = {
                  "context": contexts,
                  "question": questions,
                  "is_impossible": is_impossible,
                  "answer_start": answers_start,
                  "answer_text": answers_text
              }
              output_df = pd.DataFrame(data)
          return output_df
```

Рисунок 5.1.5 Функція для перетворення JSON у DataFrame

Отримано наступний результат:

```
[51]: display(train_df.head(2))
      display(test_df.head(2))
```

	context	question	is_impossible	answer_start	answer_text
0	Російська (ру́сский язы́к, російський язик, ви...	Де російська мова є офіційною мовою?	False	134	Росії, Білорусі, Казахстані, Киргизії та багат...
1	Російська (ру́сский язы́к, російський язик, ви...	Де російська популярна, але не офіційна мова?	False	262	Україні, Латвії, Естонії

	context	question	is_impossible	answer_start	answer_text
0	Джексонвілл - найбільше місто за кількістю нас...	У якому місті Флорида найбільше населення?	False	0	Джексонвілл
1	Джексонвілл - найбільше місто за кількістю нас...	У якому місті Флорида найбільше населення?	False	0	Джексонвілл

Рисунок 5.1.6 Тренувальний та тестовий набори даних

Можна спостерігати, наші набори даних містять колонку “context” що містить контекст для пошуку запитання, колонку “question”, що містить запитання, а також колонки “answer_start” та “answer_text”, що містять початок відповіді на запитання у тексті та саму відповідь відповідно. Для створення текстових кодерів для перетворення даного набору даних на вхідний створено

колонку “answer_end”, куди була занесена інформація про останній символ відповіді.

[29]:
```
test_df['answer_end'] = 0
train_df['answer_end'] = 0
```

[30]:
```
def process_row(row):

    idx, data = row
    real_answer = data['answer_text']
    start_idx = data['answer_start']

    modified_data = data.copy()  # Create a new dictionary

    if isinstance(start_idx, list):
        return idx, modified_data

    end_idx = start_idx + len(real_answer)

    if data['context'][start_idx:end_idx] == real_answer:
        modified_data['answer_end'] = end_idx
    elif data['context'][start_idx - 1:end_idx - 1] == real_answer:
        modified_data['answer_start'] = start_idx - 1
        modified_data['answer_end'] = end_idx - 1
    elif data['context'][start_idx - 2:end_idx - 2] == real_answer:
        modified_data['answer_start'] = start_idx - 2
        modified_data['answer_end'] = end_idx - 2

    return idx, modified_data
```

Рисунок 5.1.7 Функція для створення колонки з індексом останнього символу відповіді

[31]:
```
def create_end_idx(df):
    num_processes = multiprocessing.cpu_count()
    pool = multiprocessing.Pool(processes=num_processes)

    rows = list(df.iterrows())  # Convert iterrows() to a list
    # rows = df.iterrows()
    # display(rows)
    result = pool.map(process_row, rows)  # Pass the list to pool.map()
    # result = pool.map(process_row, df.iterrows())
    processed_rows = [(idx, data) for idx, data in result]

    pool.close()
    pool.join()

    output_df = pd.DataFrame([data for idx, data in processed_rows], index=[str(idx) for idx, data in processed_rows], columns=df.column
    return output_df

start_time = time.time()
train_df_with_end = create_end_idx(train_df)
test_df_with_end = create_end_idx(test_df)
end_time = time.time()
print(abs(start_time - end_time))
```

```
59.04617977142334
```

Рисунок 5.1.8 Продовження попередньої частини з використанням розпаралелювання

Далі зазначено назву моделі, яка буде дотреновуватись. Для отримання натренованої версії моделі XLM-roBERTa було використано сервіс HuggingFace, з його хабом з моделями. За допомогою даного сервісу можна завантажити майже будь-яку необхідну модель, яка попередньо буде навченою і готовою до роботи. У фрагменті коду нижче можна спостерігати за завантаженням токенайзера для моделі, який автоматично перетворить набір даних у набір токенів, зрозумілих моделі.

```
[38]: tokenizer = AutoTokenizer.from_pretrained(tokenizer_name)

      train_encodings = tokenizer(train_df_with_end['context'].values.tolist(),\
                                  train_df_with_end['question'].values.tolist(),\
                                  truncation=True, padding=True)
      val_encodings = tokenizer(test_df_with_end['context'].values.tolist(),\
                                test_df_with_end['question'].values.tolist(),\
                                truncation=True, padding=True)
```

```
Downloading:   0%|          | 0.00/179 [00:00<?, ?B/s]
Downloading:   0%|          | 0.00/606 [00:00<?, ?B/s]
Downloading:   0%|          | 0.00/5.07M [00:00<?, ?B/s]
Downloading:   0%|          | 0.00/150 [00:00<?, ?B/s]
```

Рисунок 5.1.9 Створення тренувального та валідаційного кодерів

Для того, щоб перетворити колонки даних “answer_start” та “answer_end” в дані, що необхідні для тренування, потрібно передати їх в кодер як “start_position” та “end_position”.

```
[40]: test_df_only_possible = test_df_with_end.dropna(subset=['answer_start']).reset_index(drop=True)
      print(test_df_only_possible.is_impossible.value_counts())

      train_df_only_possible = train_df_with_end.dropna(subset=['answer_start']).reset_index(drop=True)
      print(train_df_only_possible.is_impossible.value_counts())
```

```
False    17398
Name: is_impossible, dtype: int64
False    112685
Name: is_impossible, dtype: int64
```

```
[41]: train_df_only_possible[['answer_start', 'answer_end']] = train_df_only_possible[['answer_start', 'answer_end']].astype(int)

      test_df_only_possible[['answer_start', 'answer_end']] = test_df_only_possible[['answer_start', 'answer_end']].astype(int)
```

```
[42]: test_df_only_possible['answers'] = test_df_only_possible.apply(lambda row: {
          'answer_text': row['answer_text'],
          'answer_start': row['answer_start'],
          'answer_end': row['answer_end']
      }, axis=1)

      train_df_only_possible['answers'] = train_df_only_possible.apply(lambda row: {
          'answer_text': row['answer_text'],
          'answer_start': row['answer_start'],
          'answer_end': row['answer_end']
      }, axis=1)
```

Рисунок 5.1.10 попередня обробка даних

```
[43]: def add_token_positions(encodings, answers):
          # initialize lists to contain the token indices of answer start/end
          start_positions = []
          end_positions = []

          for idx in range(len(answers)):
              start_positions.append(encodings.char_to_token(idx, answers[idx]['answer_start']))

              end_positions.append(encodings.char_to_token(idx, answers[idx]['answer_end']))

              if start_positions[-1] is None:
                  start_positions[-1] = tokenizer.model_max_length-1
              if end_positions[-1] is None:
                  end_positions[-1] = tokenizer.model_max_length-1

          # update our encodings object with the new token-based start/end positions
          encodings.update({'start_positions': start_positions, 'end_positions': end_positions})

[44]: # apply function to our data
      add_token_positions(train_encodings, train_df_only_possible['answers'])
      add_token_positions(val_encodings, test_df_only_possible['answers'])
```

Рисунок 5.1.11 Перетворення колонок answer_start та answer_end в start_position та end_position.

Після попередньої обробки даних відбувається їх завантаження на сервери S3 для подальшого тренування:

```
# Upload the serialized encodings to S3
s3_client.put_object(Body=serialized_encodings_train, Bucket=sagemaker_session_bucket, Key=file_path_train)
s3_client.put_object(Body=serialized_encodings_test, Bucket=sagemaker_session_bucket, Key=file_path_test)

print("Encodings saved to S3 successfully.")
```

Рисунок 5.1.12 Перетворення колонок “answer_start” та “answer_end” в “start_position” та “end_position”.

Наступними кроками розробки є створення HuggingFace estimator та початок тренування моделі.

HuggingFace Estimator спрощує процес навчання та розгортання моделей NLP, надаючи високорівневий інтерфейс до базової інфраструктури та ресурсів, необхідних для навчання. Він абстрагується від складнощів налаштування навчального середовища, управління вхідними даними та розгортання моделі.

HuggingFace Estimator використовує можливості розподіленого навчання SageMaker, дозволяючи вам ефективно навчати моделі NLP на великих наборах даних у декількох екземплярах. Він автоматично масштабує навчальні ресурси на основі заданої вами конфігурації, що полегшує роботу з великомасштабними навчальними завданнями NLP.

Після завершення навчання HuggingFace Estimator полегшує розгортання моделі за допомогою хостингової інфраструктури SageMaker. Це спрощує процес обслуговування навченої моделі як веб-сервісу та дозволяє вам робити прогнози в режимі реального часу за допомогою розгорнутої NLP-моделі.

Нижче представлено створення назви для процесу навчання:

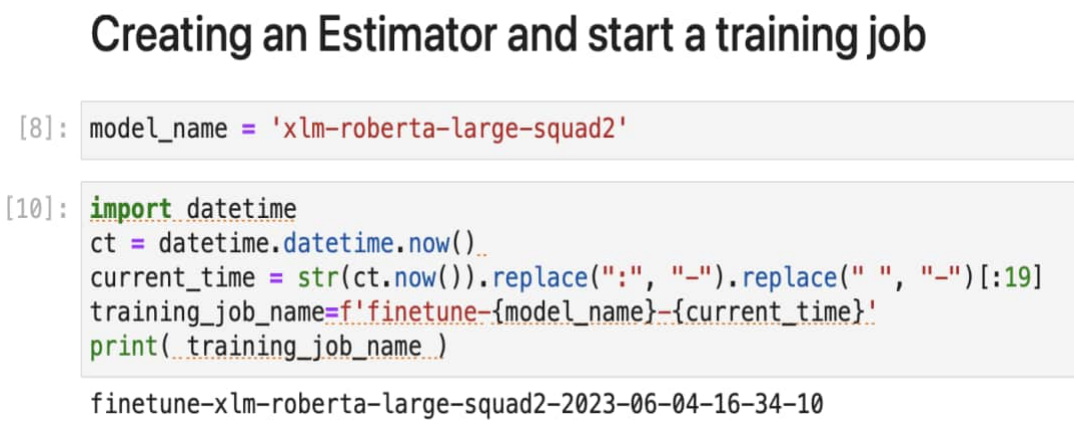

Рисунок 5.1.13 Перетворення колонок answer_start та answer_end в start_position та end_position.

Далі перераховано гіперпараметри, що будуть використовуватися для тренуванні моделі, та передаватися в HuggingFace Estimator:

```
[11]: hyperparameters={'epochs': 3,
                       'train_batch_size': 32,
                       'model_name': model_name,
                       'tokenizer_name': tokenizer_name,
                       'output_dir':'/opt/ml/checkpoints',
                       }
```

Рисунок 5.1.14 Зазначення гіперпараметрів тренування

А також вказано метрики, що будуть використовуватися під час тренування для корекції моделі:

```
[12]: metric_definitions=[
        {'Name': 'loss', 'Regex': "'loss': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'learning_rate', 'Regex': "'learning_rate': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'eval_loss', 'Regex': "'eval_loss': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'eval_accuracy', 'Regex': "'eval_accuracy': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'eval_f1', 'Regex': "'eval_f1': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'eval_precision', 'Regex': "'eval_precision': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'eval_recall', 'Regex': "'eval_recall': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'eval_runtime', 'Regex': "'eval_runtime': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'eval_samples_per_second', 'Regex': "'eval_samples_per_second': ([0-9]+(.|e\-)[0-9]+),?"},
        {'Name': 'epoch', 'Regex': "'epoch': ([0-9]+(.|e\-)[0-9]+),?"}]
```

Рисунок 5.1.15 зазначення метрик для тренування

Переносимо всі, попередньо заповнені дані у HuggingFace Estimator та переходимо до його створення:

```
huggingface_estimator = HuggingFace(
    entry_point='train.py',
    source_dir='./scripts',
    instance_type='ml.c5.xlarge',
    instance_count=1,
    checkpoint_s3_uri=f's3://{sagemaker_session_bucket}/models/checkpoints',
    use_spot_instances=True,
    role=role,
    transformers_version='4.26.0',
    pytorch_version='1.13.1',
    py_version='py39',
    hyperparameters = hyperparameters,
    metric_definitions=metric_definitions,
    max_run=36000, # expected max run in seconds
)
```

Рисунок 5.1.16 Створення об’єкту HuggingFace estimator для тренування

В тіло об’єкта HuggingFace передаємо аргументи, такі як:

- entry_point: передача тренувального файлу у функцію
- source_dir: шлях до тренувального файлу
- instance_type: тип інстансу, на якому буде тренуватись модель
- instanse_counts: кількість інстансів
- checkpoint_s3_url: посилання на шлях, куди будуть завантажуватись результати моделі
- use_spot_instances: Вказує, чи використовувати екземпляри SageMaker Managed Spot для навчання та інші.

Тренувальний файл, який використовується в тренувальній функції має наступний вигляд:

```
"""
Training script for Hugging Face SageMaker Estimator
"""
import logging
import sys
import argparse
import os
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from transformers import Trainer, TrainingArguments
from datasets import load_from_disk
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

if __name__ == "__main__":

    parser = argparse.ArgumentParser()

    # hyperparameters sent by the client are passed as command-line arguments to the script.
    parser.add_argument("--epochs", type=int, default=3)
    parser.add_argument("--train_batch_size", type=int, default=32)
    parser.add_argument("--eval_batch_size", type=int, default=64)
    parser.add_argument("--warmup_steps", type=int, default=500)
    parser.add_argument("--model_name", type=str)
    parser.add_argument("--tokenizer_name", type=str)
    parser.add_argument("--learning_rate", type=str, default=5e-5)

    # Data, model, and output directories
    parser.add_argument("--output-data-dir", type=str, default=os.environ["SM_OUTPUT_DATA_DIR"])
    parser.add_argument("--model-dir", type=str, default=os.environ["SM_MODEL_DIR"])
    parser.add_argument("--n_gpus", type=str, default=os.environ["SM_NUM_GPUS"])
    parser.add_argument("--training_dir", type=str, default=os.environ["SM_CHANNEL_TRAIN"])
    parser.add_argument("--test_dir", type=str, default=os.environ["SM_CHANNEL_TEST"])

    args, _ = parser.parse_known_args()

    # Set up logging
    logger = logging.getLogger(__name__)

    logging.basicConfig(
        level=logging.getLevelName("INFO"),
        handlers=[logging.StreamHandler(sys.stdout)],
        format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
    )

    # load datasets
    train_dataset = load_from_disk(args.training_dir)
    test_dataset = load_from_disk(args.test_dir)

    logger.info("loaded train_dataset length is: %s", len(train_dataset))
    logger.info("loaded test_dataset length is: %s", len(test_dataset))
```

Рисунок 5.1.17 Тіло тренувального файлу. Частина перша

```python
def compute_metrics(pred):
    """Compute metrics function for binary classification"""
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    precision, recall, f_1, _ = precision_recall_fscore_support(labels, preds, average="binary")
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc, "f1": f_1, "precision": precision, "recall": recall}

# download model and tokenizer from model hub
model = AutoModelForSequenceClassification.from_pretrained(args.model_name)
tokenizer = AutoTokenizer.from_pretrained(args.tokenizer_name)


# define training args
training_args = TrainingArguments(
    output_dir=args.model_dir,
    num_train_epochs=args.epochs,
    per_device_train_batch_size=args.train_batch_size,
    per_device_eval_batch_size=args.eval_batch_size,
    warmup_steps=args.warmup_steps,
    evaluation_strategy="epoch",
    logging_dir=f"{args.output_data_dir}/logs",
    learning_rate=float(args.learning_rate),
)

# create Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=tokenizer,
)

# train model
trainer.train()

# evaluate model
eval_result = trainer.evaluate(eval_dataset=test_dataset)

# writes eval result to file which can be accessed later in s3 ouput
with open(os.path.join(args.output_data_dir, "eval_results.txt"), "w") as writer:
    print("***** Eval results *****")
    for key, value in sorted(eval_result.items()):
        writer.write(f"{key} = {value}\n")

# Saves the model to s3
trainer.save_model(args.model_dir)
```

Рисунок 5.1.18 Тіло тренувального файлу. Частина друга

У файлі наведеному вище відбувається ініціалізація гіперпараметрів моделі, виведення метрик, створення конфігураційних файлів, створення тренувальної роботи та запис результатів тренування у файл.

Щоб розпочати донавчання, необхідно викликати функцію fit(), і після цього Sagemaker автоматично створить процес тренування, і сповістить про його закінчення.

```
Train Epoch: 1 [0/5709 (0%)] Loss: 0.752307
INFO:__main__:Train Epoch: 1 [0/5709 (0%)] Loss: 0.752307
Train Epoch: 1 [3200/5709 (56%)] Loss: 0.518363
INFO:__main__:Train Epoch: 1 [3200/5709 (56%)] Loss: 0.518363
Average training loss: 0.473186
INFO:__main__:Average training loss: 0.473186
Test set: Accuracy: 0.825221
INFO:__main__:Test set: Accuracy: 0.825221Train Epoch: 2 [0/5709
(0%)] Loss: 0.367052
INFO:__main__:Train Epoch: 2 [0/5709 (0%)] Loss: 0.367052
Train Epoch: 2 [3200/5709 (56%)] Loss: 0.313773
INFO:__main__:Train Epoch: 2 [3200/5709 (56%)] Loss: 0.313773
Average training loss: 0.356163
INFO:__main__:Average training loss: 0.356163
Test set: Accuracy: 0.824115Saving tuned model.
INFO:__main__:Test set: Accuracy: 0.824115
INFO:__main__:Saving tuned model.
```

Рисунок 5.1.19 Журнал з файлу для тренування

Після отримання та аналізу результатів, можна приступати до запуску моделі як сервісу. Для цього ми викликаємо функцію deploy():

Deployment

```
]: predictor = huggingface_estimator.deploy(initial_instance_count=1, instance_type="ml.m5.xlarge", endpoint_name=training_job_name)
```

Рисунок 5.1.20 Запуск системи як сервісу

5.2Модель як API сервіс

AWS Lambda - це сервіс без серверних обчислень, що надається Amazon Web Services (AWS). Він дає змогу запускати код без резервування або керування серверами. З AWS Lambda можна зосередитися на написанні логіки програми, а AWS подбає про базову інфраструктуру, масштабування та доступність. Цей сервіс використовується як міст між моделлю та AWS Gateway API, що перетворює модель в API сервіс. [19]

Як можна бачити на Рис 5.2.1, AWS Lambda з’єднує відповідь моделі та API Gateway сервіс та стає обгорткою для моделі.

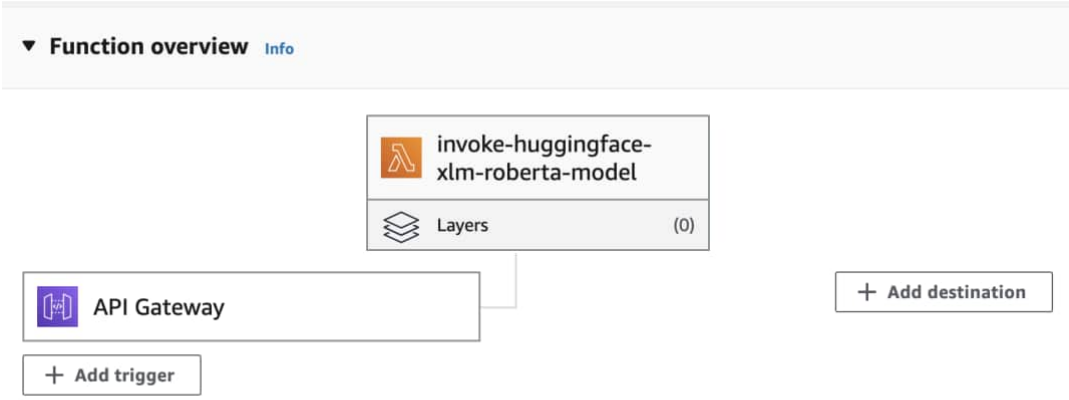

Рисунок 5.2.1 Схема роботи Lambda-фукнції

Нижче представлено тіло файлу lambda_function.py, де розміщена головна логіка роботи даної функції. Вона має декілька перевірок на коректність запитів, а також повертає різний результат в залежності від типу запиту:

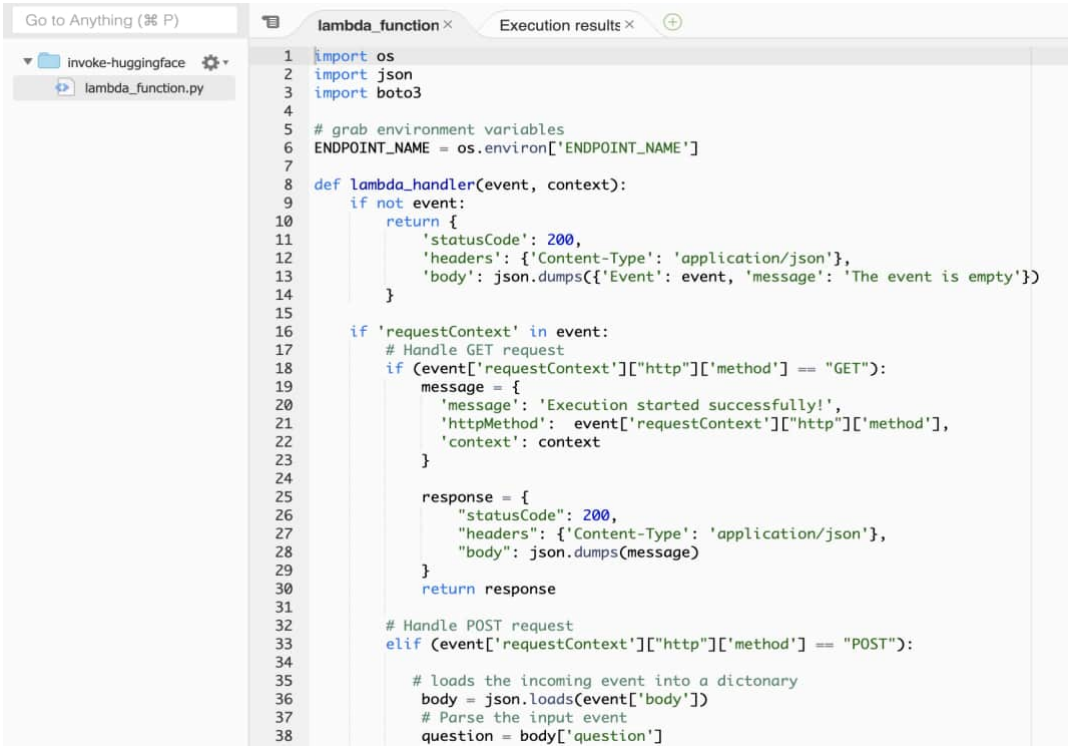

Рисунок 5.2.2 Файл, що використовується в Lambda функції для її коректної роботи. Частина перша

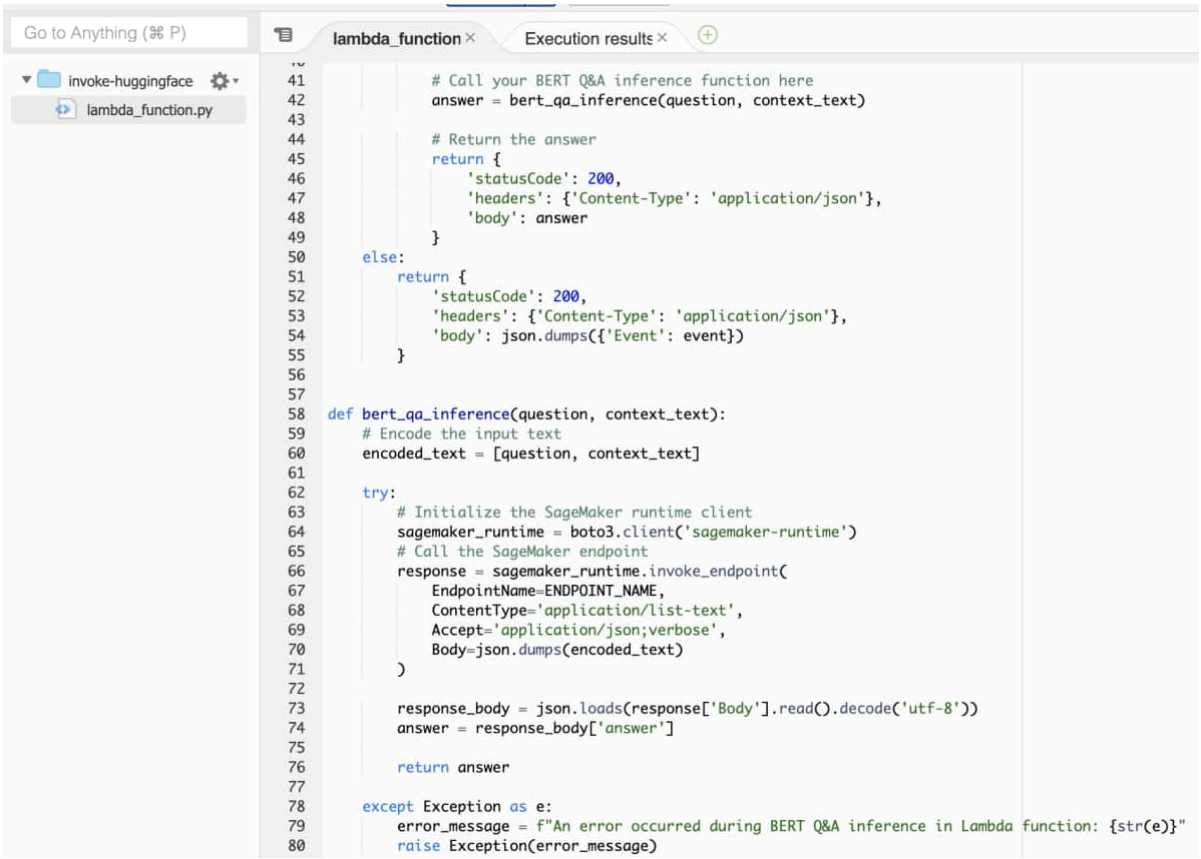

Рисунок 5.2.3 Файл, що використовується в Lambda-функції для її коректної роботи. Частина друга

В інтерфейсі самої Lambda-функції також є місце для функції тестування. Її було додано, щоб можна було чітко виявити проблеми вже на етапі розробки. Один з тестів містять в собі перевірку звичного режиму роботи моделі, тобто відповіді на питання. Тому перезапускаємо його, і дивимось на відповідь системи. До прикладу ставимо таке запитання«Яке місто є столицею Франції?», і надаємо частину тексту з Вікіпедії. Запустивши наш Lambda-додаток, та отримавши коректну відповідь, ще раз впевнюємось що все працює.

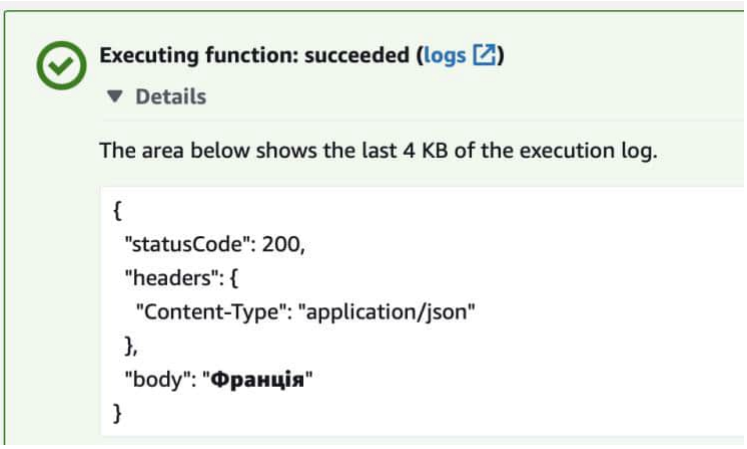

Рисунок 5.2.4 Приклад успішно пройденого тесту

Рисунок 5.2.5 Частина тесту, результат якого ви можете бачити на попередній сторінці

Важливим наступним компонентом є з’єднання нашої Lambda-функції з сервісів API Gateway, що забезпечить нам легкий спосіб використання нашої моделі як API.

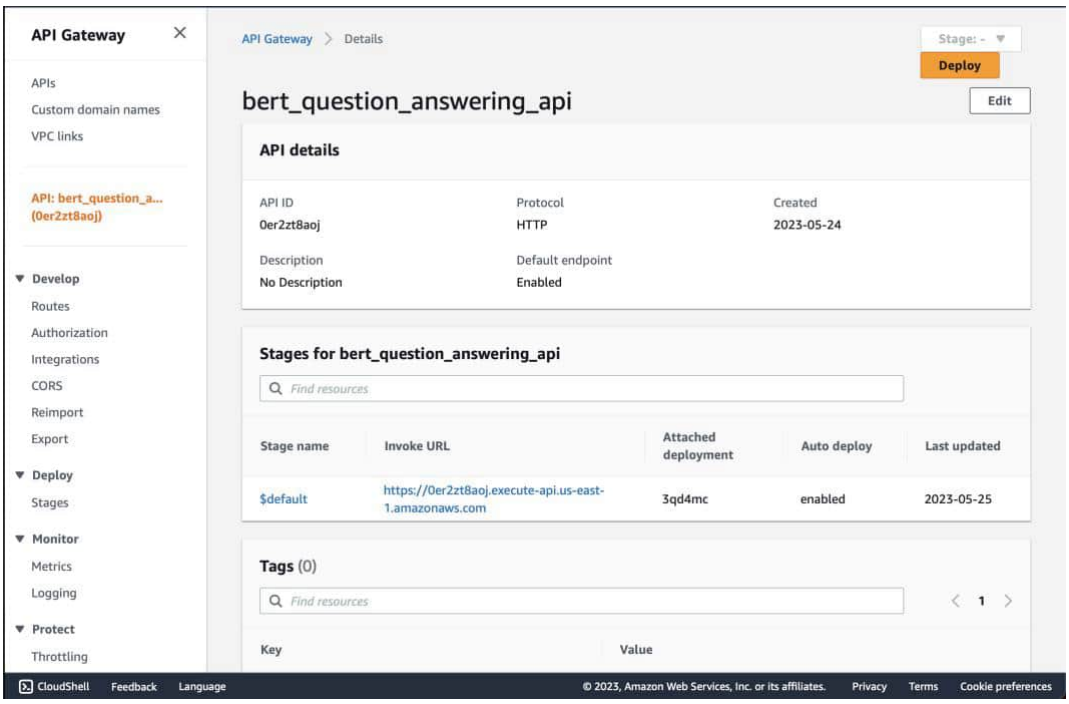

Рисунок 5.2.6 Інтерфейс платформи API Gateway

Для того, щоб ретельно перевірити роботу AWS Gateway API необхідно згенерувати декілька запитів у додатку Postman, що використовується для перевірки вебзастосунківта API.

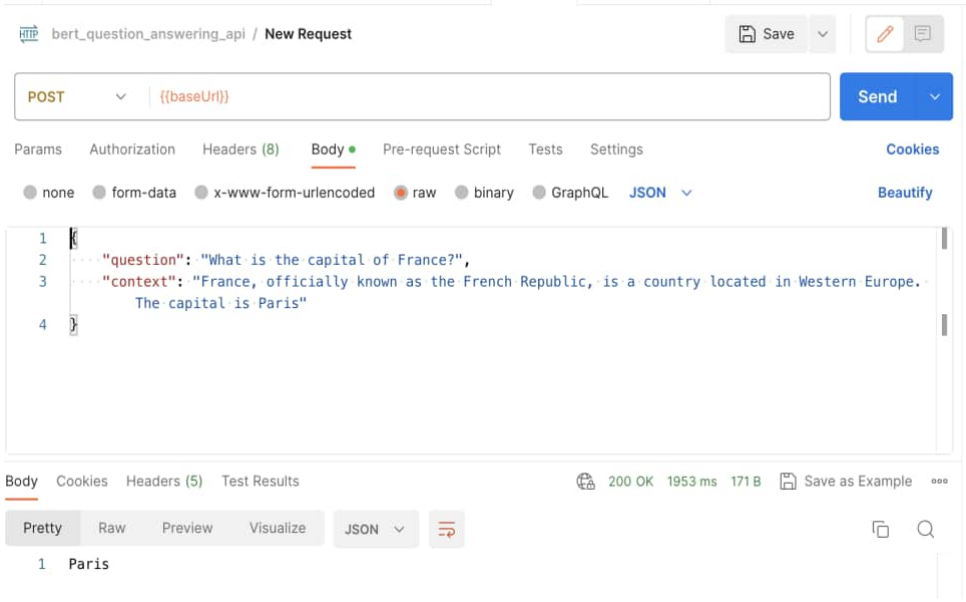

Рисунок 5.2.6 Інтерфейс платформи Postman та результат роботи нашої моделі на запитанні

На рисунку 5.2.6 видно, що сервіс працює без будь-яких проблем, і можна розпочинати з'єднання його з вебзастосунком.

5.3Створення вебзастосунку

Для створення вебдодатку для цієї системи було обрано Flask. Flask - це популярний мікро вебфреймворк, написаний на Python. Він простий та зручний у використанні та надає розробникам необхідні інструменти для швидкого та ефективного створення вебдодатків.

Використання Flask для створення веб-додатків має кілька переваг:

Легкість та мінімалізм: Він надає лише основні інструменти та функції, необхідні для створення вебдодатків, без зайвих накладних витрат. Ця простота робить Flask легким у вивченні, використанні та підтримці.

Гнучкість: Flask дає розробникам можливість гнучко структурувати свої додатки так, як вони вважають за потрібне. Він не нав'язує жорсткої структури проєкту і не нав'язує певних архітектурних шаблонів. Це дає змогу розробникам мати більше контролю та свободи при розробці своїх додатків на основі їхніх конкретних потреб та вподобань.

Масштабованість та модульність: Flask дотримується модульного підходу до проєктування, що дає змогу розробникам додавати або видаляти компоненти відповідно до вимог їхніх додатків. Він надає широкий спектр розширень і бібліотек, відомих як розширення Flask, які можна легко інтегрувати у ваш додаток. Серед них є розширення для підключення до бази даних, розширення для автентифікації, розширення кешування та інші.

Інтеграція з Python: Flask створено з використанням Python, це робить Flask звичним інструментом для Python-розробників, оскільки вони можуть

використовувати наявні знання та екосистему Python. За допомогою Flask ви можете легко інтегрувати бібліотеки та пакети Python у ваш вебдодаток, що робить його дуже універсальним і здатним вирішувати складні завдання.

Файл app.py відповідає за об’єднання елементів на екрані з функціоналом:

app.py > ...

```python
from flask import Flask, render_template, jsonify, request, flash, redirect
from werkzeug.datastructures import FileStorage
from werkzeug.utils import secure_filename
from fileinput import filename
import config
import model_api
import os


def page_not_found(e):
    return render_template("404.html"), 404


ALLOWED_EXTENSIONS = set(["pdf", "txt", "docx"])


def allowed_file(filename):
    return "." in filename and filename.rsplit(".", 1)[1].lower() in ALLOWED_EXTENSIONS


app = Flask(__name__)
app.secret_key = "31242"

path = os.getcwd()
# file Upload
UPLOAD_FOLDER = os.path.join(path, "data")
if not os.path.isdir(UPLOAD_FOLDER):
    os.mkdir(UPLOAD_FOLDER)
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

app.register_error_handler(404, page_not_found)
```

Рисунок 5.3.1 Файл app.py. Перша частина

```python
@app.route("/upload", methods=["POST"])
def upload_file():
    if request.method == "POST":
        if "file" not in request.files:
            flash("No file part")
            return redirect(request.url)

        context_file = request.files["file"]

        if context_file.filename == "":
            flash("No file selected for uploading")
            return redirect(request.url)

        if context_file and allowed_file(context_file.filename):
            filename = secure_filename(context_file.filename)
            context_file.save(os.path.join(app.config["UPLOAD_FOLDER"], filename))
            flash("File successfully uploaded")
            return redirect("/")
        else:
            flash("Allowed file types are txt, pdf, png, jpg, jpeg, gif")
            return redirect(request.url)
```

Рисунок 5.3.2 Файл app.py, друга частина. Функція для завантажування документів у додаток

```python
@app.route("/", methods=["POST", "GET"])
def index():
    if request.method == "POST":
        question = request.form["question"]
        context = request.form["context"]

        res = {}
        res["answer"] = model_api.getModelAPIResponse(question, context)
        return jsonify(res), 200
    return render_template("index.html", **locals())


if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8888", debug=True)

```

Рисунок 5.3.3 Файл app.py, третя частина. Функція для виклику API для отримання відповіді на запитання.

Далі йде файл, через який можна отримати доступ до нашої моделі через API

model_api.py > getModelAPIResponse

```python
import config
import requests
import json

# call the aws Gateway API to get the response
def getModelAPIResponse(question:str, context:str) -> str:
    url = "xxxxxxxxxxxxxxxxxxx"

    payload = json.dumps({
        "question": question,
        "context": context
    })
    headers = {
        'Content-Type': 'application/json'
    }

    response = requests.request("POST", url, headers=headers, data=payload)

    return response.text
```

Рисунок 5.3.4 Файл model_api.py, використовується для доступу до API

А також файли, де зберігається весь вигляд сайту:

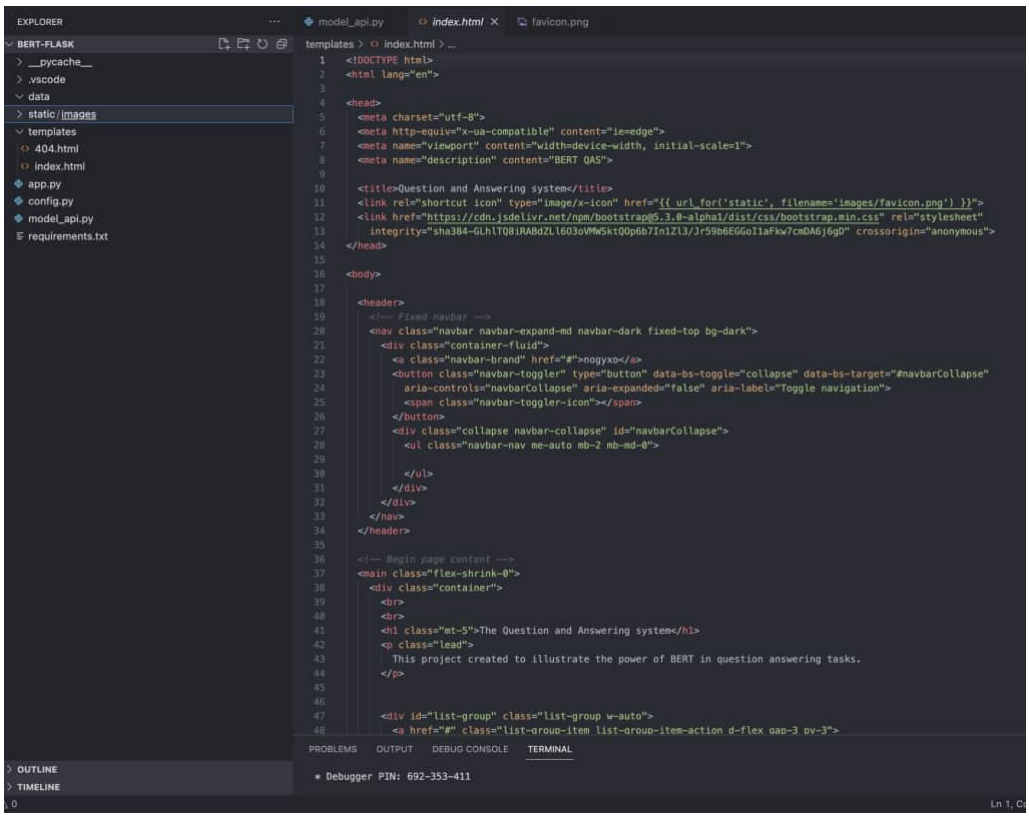

Рисунок 5.3.5 Файл головної сторінки сайту. Відповідає за відображення елементів на екрані

Після запуску файлу app.py, з’являється моливість доступитись локально до вебсайту:

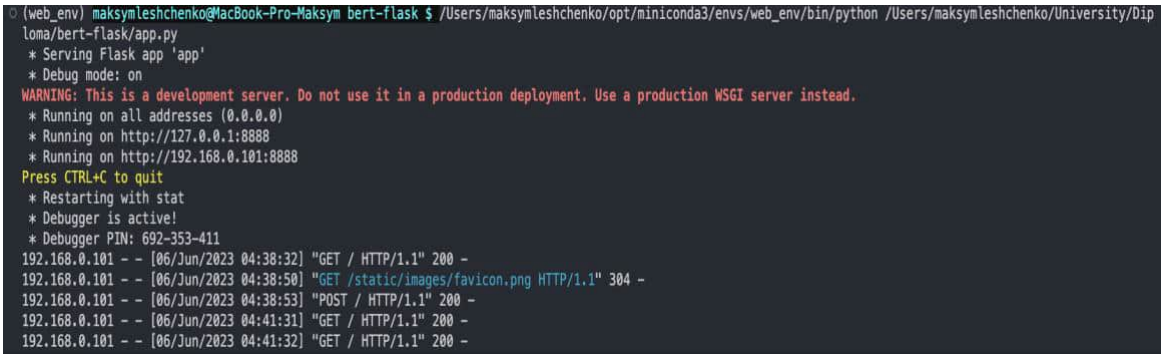

Рисунок 5.3.6 Запуск вебзастосунку

Після запуску, перейшовши за посиланням, отримуємо доступ до сайту, та тестуємо роботу моделі:

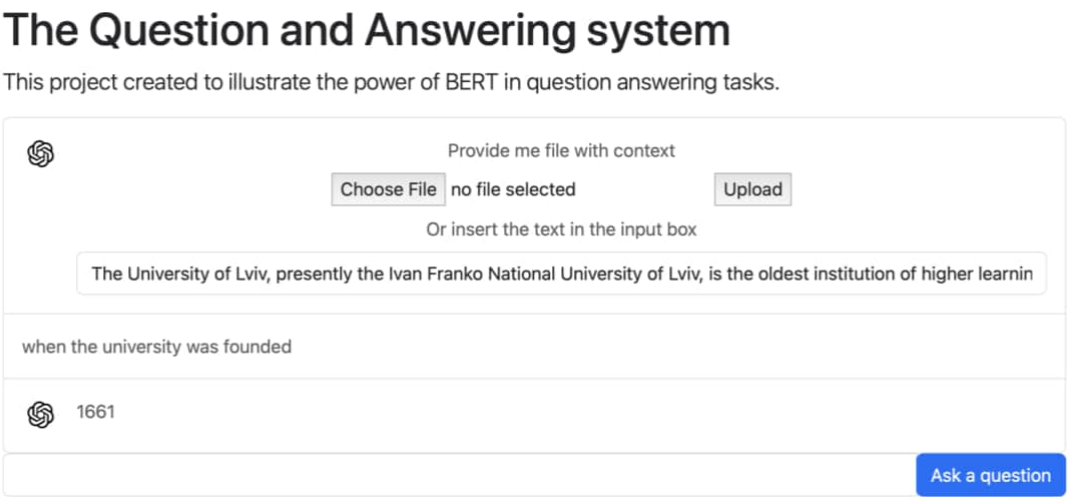

Рисунок 5.3.7 Результат відповіді на питання у нашому вебзастосунку

ВИСНОВОК

Підсумовуючи, створення вебдодатку з використанням системи запитань і відповідей (QA), побудованої за допомогою BERT (Bidirectional Encoder Representations from Transformers), продемонструвало значний потенціал у покращенні користувацького досвіду та пошуку інформації. Використання BERT, найсучаснішої мовної моделі, дає змогу системі запитань та відповідей і генерувати точні відповіді на запити користувачів.

Використовуючи контекстне розуміння BERT і можливості глибокого навчання, система запитань та відповідей може обробляти вхідні дані природною мовою і надавати точні відповіді, що призводить до підвищення задоволеності користувачів і ефективності роботи. Здатність моделі фіксувати складні взаємозв'язки та контекстуальні нюанси в тексті дає змогу їй досягати успіху в різних сферах, включаючи підтримку клієнтів, платформи для обміну знаннями та контенто-орієнтовані додатки.

Інтеграція систем запитань та відповідей на основі BERT у вебдодатки має кілька переваг. Вона дає змогу користувачам взаємодіяти з додатком, ставлячи запитання та отримуючи релевантні відповіді, підвищуючи їхню зацікавленість і полегшуючи пошук інформації. Здатність системи обробляти складні запити та надавати точні відповіді підвищує ефективність і економить час користувачів, що призводить до покращення користувацького досвіду.

Крім того, гнучкість і розширюваність BERT дають змогу розробникам тонко налаштовувати модель на наборах даних конкретної галузі, пристосовуючи її до конкретних вимог програми. Така адаптивність гарантує, що система запитань та відповідей може надавати точні та релевантні домену відповіді, незалежно від предметної області.

Хоча при розгортанні та підтримці системи запитання та відповідей на основі BERT у вебдодатку можуть виникати певні труднощі, такі як розмір моделі та обчислювальні ресурси, переваги переважають ці перешкоди. Постійний розвиток хмарних обчислень та інфраструктури в поєднанні з наявністю попередньо навчених BERT-моделей і фреймворків, таких як Hugging Face, спростили процес інтеграції та зробили його більш доступним для розробників.

Таким чином, інтеграція системи запитань та відповідей на основі BERT у вебдодатки є перспективним шляхом для покращення користувацького досвіду та пошуку інформації. Поєднання можливостей розуміння мови BERT та інтерактивності вебдодатків створює потужний інструмент для ефективного доступу та обробки інформації. З подальшими досягненнями в обробці природної мови та глибокому навчанні ми можемо очікувати ще більш досконалих систем запитань та відповідей, які революціонізують спосіб взаємодії з вебдодатками та доступу до знань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Attention is all you need by Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia, URL: https://arxiv.org/pdf/1706.03762.pdf (Дата звернення: 07.05.2023)
2. Michael Caballero, A Brief Survey of Question Answering Systems, 2021. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3996229 (Дата звернення: 09.05.2023)
3. Kalyanpur, A., Patwardhan, S., Boguraev, B.K., Lally, A. and Chu-Carroll, J., 2012. Fact-based question decomposition in DeepQA. IBM Journal of Research and Development, 56(3.4), pp.13–1. URL: https://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=6177717 (Дата звернення: 11.05.2023)
4. Peter Bloem, “Transformers from scratch” blog post, 2019. URL: https://peterbloem.nl/blog/transformers (Дата звернення: 01.05.2023)
5. Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” URL: https://arxiv.org/abs/1810.04805 (Дата звернення: 09.05.2023)
6. Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. 2020. What Happens To BERT Embeddings During Fine-tuning?. In Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 33–44, Online. Association for Computational Linguistics. URL: https://aclanthology.org/2020.blackboxnlp-1.4/ (Дата звернення: 12.05.2023)
7. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. URL: https://arxiv.org/abs/1907.11692 (Дата звернення: 10.05.2023)

8. Serhii TIUTIUNNYK Vsevolod DYOMKIN, Context-Based Question-Answering System for the Ukrainian Language URL: https://er.ucu.edu.ua/bitstream/handle/1/1898/Tiutiunnyk_Context-based%20Question-answering.pdf?sequence=1&isAllowed=y (Дата звернення: 12.05.2023)
9. AWS SageMaker: Train, Deploy and Update a Hugging Face BERT Model https://medium.com/analytics-vidhya/aws-sagemaker-train-deploy-and-update-a-hugging-face-bert-model-eeefc8211368 (Дата звернення: 20.05.2023)
10. Baseball: an automatic question-answerer by Bert F. Green, Alice K. Wolf, Carol Chomsky, Kenneth Laughery. URL: https://dl.acm.org/doi/abs/10.1145/1460690.1460714 (Дата звернення: 07.05.2023)
11. A literature review on question answering techniques, paradigms and systems by Marco Antonio Calijorne Soares, Fernando Silva Parreiras. URL: https://www.sciencedirect.com/science/article/pii/S131915781830082X?via%3Dihub (Дата звернення: 10.05.2023)
12. R-NET: Machine Reading Comprehension With Self-Matching Networks∗ by Natural Language Computing Group, Microsoft Research Asia. P.6-7, URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf (Дата звернення: 11.05.2023)
13. FusionNet: Fusing via Fully-Aware Attention with Application to Machine Comprehension by Hsin-Yuan Huang*1,2, Chenguang Zhu1, Yelong Shen1, Weizhu Chen1 URL: https://paperswithcode.com/paper/fusionnet-fusing-via-fully-aware-attention/review/ (Дата звернення: 11.05.2023)
14. XLNet: Generalized Autoregressive Pretraining for Language Understanding by Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le. URL: https://arxiv.org/abs/1906.08237. (Дата звернення: 23.05.2023)
15. Unsupervised Cross-lingual Representation Learning at Scale by Alexis Conneau∗, Kartikay Khandelwal∗, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzma´n, Edouard Grave, Myle Ott, Luke Zettlemoyer, Veselin Stoyanov. URL: https://arxiv.org/pdf/1911.02116.pdf (Дата звернення: 11.05.2023)
16. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter by Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf. ULR: https://arxiv.org/abs/1910.01108 (Дата звернення: 14.05.2023)

17. Know What You Don't Know: Unanswerable Questions for SquAD by Pranav Rajpurkar, Robin Jia, Percy Liang. URL: https://arxiv.org/abs/1806.03822 (Дата звернення: 14.05.2023)
18. Сторінка Amazon web services. URL: https://aws.amazon.com/?nc2=h_lg (Дата звернення: 14.05.2023)
19. Сторінка документації AWS Lambda URL: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (Дата звернення: 14.05.2023)

ДОДАТОК А

Вебдодаток Flask:
app.py file:

```
from flask import Flask, render_template, jsonify, request, flash, redirect
from werkzeug.datastructures import FileStorage
from werkzeug.utils import secure_filename
from fileinput import filename
import config
import model_api
import os

def page_not_found(e):
    return render_template("404.html"), 404

ALLOWED_EXTENSIONS = set(["pdf", "txt", "docx"])

def allowed_file(filename):
    return "." in filename and filename.rsplit(".", 1)[1].lower() in
ALLOWED_EXTENSIONS

app = Flask(__name__)
app.secret_key = "31242"

path = os.getcwd()
# file Upload
UPLOAD_FOLDER = os.path.join(path, "data")
if not os.path.isdir(UPLOAD_FOLDER):
    os.mkdir(UPLOAD_FOLDER)
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

app.register_error_handler(404, page_not_found)

@app.route("/upload", methods=["POST"])
def upload_file():
    if request.method == "POST":
        if "file" not in request.files:
            flash("No file part")
            return redirect(request.url)

        context_file = request.files["file"]

        if context_file.filename == "":
            flash("No file selected for uploading")
            return redirect(request.url)
```

```
        if context_file and allowed_file(context_file.filename):
            filename = secure_filename(context_file.filename)
            context_file.save(os.path.join(app.config["UPLOAD_FOLDER"], filename))
            flash("File successfully uploaded")
            return redirect("/")
        else:
            flash("Allowed file types are txt, pdf, png, jpg, jpeg, gif")
            return redirect(request.url)

@app.route("/", methods=["POST", "GET"])
def index():
    if request.method == "POST":
        question = request.form["question"]
        context = request.form["context"]

        res = {}
        res["answer"] = model_api.getModelAPIResponse(question, context)
        return jsonify(res), 200
    return render_template("index.html", **locals())

if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8888", debug=True)
```

Model API file:

```
import config
import requests
import json

# call the aws Gateway API to get the response
def getModelAPIResponse(question:str, context:str) -> str:
    url = "https://0er2zt8aoj.execute-api.us-east-1.amazonaws.com/"

    payload = json.dumps({
        "question": question,
        "context": context
    })
    headers = {
        'Content-Type': 'application/json'
    }

    response = requests.request("POST", url, headers=headers, data=payload)

    return response.text
```

index.html file:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="BERT QAS">

  <title>Question and Answering system</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-
GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">
</head>

<body>

  <header>
    <!-- Fixed navbar -->
    <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">nogyxo</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-
bs-target="#navbarCollapse"
          aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle
navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarCollapse">
          <ul class="navbar-nav me-auto mb-2 mb-md-0">

          </ul>
        </div>
      </div>
    </nav>
  </header>

  <!-- Begin page content -->
  <main class="flex-shrink-0">
    <div class="container">
      <br>
      <br>
```

```
    <h1 class="mt-5">The Question and Answering system</h1>
    <p class="lead">
      This project created to illustrate the power of BERT in question answering tasks.
    </p>

    <div id="list-group" class="list-group w-auto">
      <a href="#" class="list-group-item list-group-item-action d-flex gap-3 py-3">
        <img src="https://digital-
practice.ams3.cdn.digitaloceanspaces.com/static%2Fapp%2Fimg%2Fopenai-
logo.png"
          alt="twbs" width="32" height="32" class="rounded-circle flex-shrink-0">
        <div class="d-flex gap-2 w-100 align-items-center flex-column">
          <div>
            <p class="mb-0 opacity-75">Provide me file with context</p>
          </div>
          <div>
            <form id="upload-file" method="POST" enctype="multipart/form-data">
              <fieldset>
                <input name="file" type="file">
                <button id="upload-file-btn" type="button">Upload</button>
              </fieldset>
            </form>
          </div>
          <div>
            <p class="mb-0 opacity-75">Or insert the text in the input box</p>
          </div>
          <input type="text" class="form-control" id="context-input">
        </div>
      </a>
    </div>
    <div class="input-group mb-3">
      <input type="text" class="form-control" id="question-input">
      <div class="input-group-append">
        <button id="ask-button" class="btn btn-primary">Ask a question</button>
      </div>
    </div>
  </div>
 </main>

 <script src="https://code.jquery.com/jquery-3.6.3.min.js"
   integrity="sha256-
pvPw+upLPUjgMXY0G+8O0xUf+/Im1MZjXxxgOcBQBXU="
crossorigin="anonymous"></script>
 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"
```

```
    integrity="sha384-
w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7C
XvN"
    crossorigin="anonymous"></script>

  <script>

    $("#ask-button").click(function () {
      var context = $("#context-input").val();
      var question = $("#question-input").val();
      let html_data = '';
      html_data += `
      <a href="#" class="list-group-item list-group-item-action d-flex gap-3 py-3">
        <div class="d-flex gap-2 w-100 justify-content-between">
          <div>
            <p class="mb-0 opacity-75">${question}</p>
          </div>
        </div>
      </a>
      `;
      $("#question-input").val('');
      $("#list-group").append(html_data);

      //AJAX CALL TO SERVER
      $.ajax({
        type: "POST",
        url: "/",
        data: { 'context': context, 'question': question },
        success: function (data) {
          let model_data = '';
          model_data += `
            <a href="#" class="list-group-item list-group-item-action d-flex gap-3 py-3">
              <img src="https://digital-
practice.ams3.cdn.digitaloceanspaces.com/static%2Fapp%2Fimg%2Fopenai-
logo.png" alt="twbs" width="32" height="32" class="rounded-circle flex-shrink-0">
              <div class="d-flex gap-2 w-100 justify-content-between">
                <div>
                  <p class="mb-0 opacity-75">${data.answer}</p>
                </div>
              </div>
            </a>
            `;
          $("#list-group").append(model_data);
        }
      });
```

```
    $(function () {
      $('#upload-file-btn').click(function () {
        var form_data = new FormData($('#upload-file')[0]);
        $.ajax({
          type: 'POST',
          url: '/upload',
          data: form_data,
          contentType: false,
          cache: false,
          processData: false,
          success: function (data) {
            console.log('Success!');
          },
        });
      });
    });
  </script>
</body>
</html>
```

Lambda file:

```
import os
import json
import boto3

# grab environment variables
ENDPOINT_NAME = os.environ['ENDPOINT_NAME']

def lambda_handler(event, context):
    if not event:
        return {
            'statusCode': 200,
            'headers': {'Content-Type': 'application/json'},
            'body': json.dumps({'Event': event, 'message': 'The event is empty'})
        }

    if 'requestContext' in event:
        # Handle GET request
        if (event['requestContext']["http"]['method'] == "GET"):
            message = {
              'message': 'Execution started successfully!',
              'httpMethod':  event['requestContext']["http"]['method'],
```

```
            'context': context
        }

        response = {
            "statusCode": 200,
            "headers": {'Content-Type': 'application/json'},
            "body": json.dumps(message)
        }
        return response

    # Handle POST request
    elif (event['requestContext']["http"]['method'] == "POST"):

        # loads the incoming event into a dictonary
        body = json.loads(event['body'])
        # Parse the input event
        question = body['question']
        context_text = body['context']

        # Call your BERT Q&A inference function here
        answer = bert_qa_inference(question, context_text)

        # Return the answer
        return {
            'statusCode': 200,
            'headers': {'Content-Type': 'application/json'},
            'body': answer
        }
else:
    return {
        'statusCode': 200,
        'headers': {'Content-Type': 'application/json'},
        'body': json.dumps({'Event': event})
    }


def bert_qa_inference(question, context_text):
    # Encode the input text
    encoded_text = [question, context_text]

    try:
        # Initialize the SageMaker runtime client
        sagemaker_runtime = boto3.client('sagemaker-runtime')
        # Call the SageMaker endpoint
        response = sagemaker_runtime.invoke_endpoint(
```

```
            EndpointName=ENDPOINT_NAME,
            ContentType='application/list-text',
            Accept='application/json;verbose',
            Body=json.dumps(encoded_text)
        )

        response_body = json.loads(response['Body'].read().decode('utf-8'))
        answer = response_body['answer']

        return answer

    except Exception as e:
        error_message = f"An error occurred during BERT Q&A inference in Lambda
function: {str(e)}"
        raise Exception(error_message)
```