NLP Group Coursework

# Introduction

National Language Processing (NLP) is a field of AI which focuses on how computers can understand and process human language. Our project involves assigning labels to sequences of words and tokens.

**Objectives**

The main objective of this project is to develop and deploy a sequence classification model that can accurately identify and label abbreviations and their long forms within biomedical literature. This is particularly useful in fields like healthcare and biomedical research, where understanding abbreviations is essential for interpreting data and extracting knowledge.

**Background**

This project builds upon our previous coursework, which focused on understanding and implementing sequence classification, in which each group member individually conducted experiments to preprocess data, train models, and evaluate performance using different algorithms and feature extraction methods. From our previous coursework, we decided as a group to test using the RoBERTa base model, as it was the best model overall.

**Dataset**

We’re using the PLOD-CW dataset, which contains 50,000 labelled tokens from scientific literature in the biomedical domain. The tokens are annotated using the BIO tagging schema, where:

* B-AC indicates the beginning of an abbreviation
* I-AC indicates an inside token of an abbreviation
* B-LF indicates the beginning of a long form
* I-LF indicates an inside token of a long form
* B-O indicates a token that is neither an abbreviation nor a long form

**Our Approach**

* Researching model serving options to identify suitable frameworks for deploying our sequence classification model.
* Building a web service to host and serve our model.
* Testing functionality by implementing a Jupyter notebook to interact with the deployed model.
* Conducting performance tests to evaluate the model's limits and capabilities.
* Implementing basic monitoring to log user inputs and model predictions.
* Analysing errors to understand the model's weaknesses and areas for improvement.

# Task 1

**FlaskAPI**

Flask is a micro web framework written in python, generally used for building web applications. FlaskAPI extends Flask by adding a layer specifically designed for API development. It is easy to use and flexible, which makes it suitable for a project of our size. Its performance in handling high-concurrency scenarios is limited, compared to asynchronous frameworks.

**FastAPI**

FastAPI is a modern, fast web framework for building APIs with Python, It is designed for building APIs quickly and efficiently, and is faster than Flask due to its asynchronous capabilities, using ASGI. It also uses on ASGI and uses starlette framework and pydantic for data validation however it can be more difficult to use as there is more of a learning curve to familiarise yourself with asynchronous programing, making flask more favourable in this case.

**Amazon SageMaker**

Amazon SageMaker is an online platform which allows the user to build train and deploy machine learning models. It has many advantages such as wide scale application. Disadvantages include complexity in set-up and managing, and costs to use.

**Heroku**

Heroku is also a cloud platform that allows developers to create, monitor and scale applications. It is compatible with various coding languages. Disadavntages include its limited scalability.

**Selection Justification**

After evaluating the pros and cons of each model, we devided to use FlaskAPI for this coursework. Our group is already familiar with Flask, which minimises the developmenttime. This familiarity allows us to focus more on the core aspects of the project. Since we want to test our model and not deploy a large-scale production system, FlaskAPI's capabilities are sufficient. FlaskAPI’s simplicity and ease of use allow us to quickly set up the eb service and focus on integrating and testing the sequence classification model.

While FastAPI offers better performance and modern features, the additional complexity and learning curve are not justified for the scope of this project. Similarly, Amazon SageMaker and Heroku, while powerful, introduce unnecessary complexity and potential costs. FlaskAPI provides a balanced solution that meets our project's needs effectively.

# Task 2: Building a web service

**Architecture Overview**

The web service consists of the three following componets:

* API Server built using FlaskAPI handles incoming HTTP requests and routes them to the appropriatae endpoints
* The model loader loads the trained RoBERTa model into memory so that it can be used for making predictions
* The Repsonse Handler formats the predictions and sends the responses bac to the client in a structured format

**Implementation**

We implemented the API server using FlaskAPI. The key steps involved in building the web service included setting up the Flask application, loading the pre-trained RoBERTa model, defining the prediction endpoint, and handling incoming requests and outgoing responses.

**Setting up FlaskAPI**

We start by setting up the Flask application and imported the necessary libraries. FlaskAPU makes it straightforward to define routes and handle requests.

\*\*code snippet\*\*

**Loading the Model**

We next loaded the RoBERTa model and tokeniser, whcih are required to process the input text and generating predictions

\*\*code snippet\*\*

**Defining the Prediction Endpoint**

We defined an endpoint that accepts POST requests. This endpoint takes the input text tokenises it, passes it through the model, and returns the prediction.

\*\*code snippet\*\*

**Hand Requests and Responses**

The endpoint function extracts the input text from the request, tokenises it, feeds it to the model, and formats the models output into a JSON response. \*\* shall i delete? \*\*

**Running the Application**

Finally we configured the application to run on the local server. The debug mode is enabled to facilitate development and troubleshooting

\*\*code snippet\*\*

# Task 3: Functionality in Notebook

**Client Function**

To interact with the deployed model, we developed a client function in a Jupyter notebook. This function sends HTTP POST requests to the FlaskAPI server with input text and receives predictions. The goal is to ensure that the model is correctly deployed and can process requests as expected.

**Implementation Details**

This involves sending a request to the “/predict” endpoint of our FlaskAPI

Task 6: Functionality in Notebook

**CI/CD Pipeline Setup**

In this project, we set up a CI/CD (Continuous Integration and Continuous Deployment) pipeline using GitHub Actions. This pipeline automates the process of building, testing, and deploying our NLP model. Below is a detailed explanation of the steps and configurations involved:

1. **Repository Creation and Structure**:

   - Created a GitHub repository named `NLP\_CW`.

   - Structured the repository with necessary directories and files, including `app`, `model`, and `.github/workflows`.

2. **Directory and File Setup**:

   - **app/main.py**: Contains the Flask web server setup to handle requests and interact with the model for predictions.

   - **app/roberta.py**: Contains functions to load the pre-trained RoBERTa model and make predictions.

   - **requirements.txt**: Lists the project dependencies (`Flask`, `transformers`, and `torch`).

   - **.github/workflows/ci\_cd\_pipeline.yml\*\***: Defines the GitHub Actions workflow for CI/CD.

3. **GitHub Actions Workflow**:

   - **Triggers**: The workflow is triggered on pushes and pull requests to the `main` branch.

   - **Build Job**:

     - Checks out the code.

     - Sets up Python environment.

      - Installs dependencies.

      - Runs tests (placeholder for actual test commands).

   - **Deploy Job**:

     - Depends on the build job.

      - Checks out the code.

     - Sets up Python environment.

     - Installs dependencies.

      - Deploys the Flask application, making it accessible via an HTTP endpoint.

4. Commit and Push:

   - Added and committed all files to the repository.

   - Pushed the changes to the `main` branch, triggering the CI/CD pipeline.