Group 6: Natural Language Processing Coursework 2

by

ELEANOR LURGIO, OLIVER KRIEGER, SEOEUN LEE, THIRI MAY THU(ALICE)

May 20, 2024

Department of Computer Science
University of Surrey
Guildford, Surrey
England, United Kingdom
GU2 7XH

Contents

Introduction

For this project, a web application has been created and deployed using two language models for the task of named entity recognition. The selected models are SpaCy and DistilBERT, due to their high performance and F1 scores from the group's previous experiments. The web application utilises these two language models to identify abbreviations and long forms based on the abbreviation detection dataset provided [1]. SpaCy is a pre-trained model that can be adapted to various tasks and is especially good for named entity recognition, which is the task required for this project. DistilBERT is a pre-trained BERT-based model, more specifically a smaller, more efficient version of BERT. Each model has been fine-tuned to perform abbreviation and longform detection.

The models described above, perform well for their size. Even after fine-tuning they do not go above a gigabyte in size, while still keeping a good prediction result. This is optimal for a web application, where everything has to be small and efficient. While the current models are loaded locally, in the future these models can be added to a server without requiring much space.

The following sections demonstrate the successful implementation and deployment of a web application. With careful consideration of each part of the project and the limitations of resources and time, both SpaCy and DistilBERT models have been deployed on a single application, allowing users to test both models with different inputs.

1 Model Serving Options

1.1 Web Frameworks and Services

1.1.1 Flask

Flask is a lightweight framework for building web applications using Python. It is a Web Server Gateway Interface (WSGI), which means it allows web servers to pass requests to applications [2]. Described as a micro web framework due to its low level of dependency on external libraries, Flask has the advantage of being easy to set up and fast to use [3].

Flask is a great choice for building the application because of its lightweight nature. It is easy to build a simple application quickly, while allowing the flexibility to scale up later on.

A possible disadvantage of Flask is that it normally cannot handle concurrent requests, so if many requests are being made at once, it may take more time to get through each of them in turn. It could also cause issues if there is a bottleneck situation, where one request is large and causes other requests to wait until it is finished. However, it is possible to reconfigure Flask to use threads to handle concurrent requests, which would avoid the issue.

1.1.2 Nginx

Nginx is an open-source web server software that excels when used for high-traffic websites due to its ability to handle many simultaneous connections [4]. While it has high performance and low resource usage, making it efficient and scalable, this comes at a cost of higher resources usage and more space required on the server.

1.1.3 Gunicorn

Like Flask, Gunicorn is a Python WSGI, popular for making web applications. The key difference is that Gunicorn is better suited for large scale applications due to its ability to handle more traffic and larger quantities of requests. Because of this ability, it means that Gunicorn is more memory intensive, as it requires more memory to run its multiple worker processes and extra features.

1.2 Deployment Services

1.2.1 AWS

AWS is a platform used by businesses and software developers. In terms of artificial intelligence, it can be used by developers to create, train and deploy their models on the cloud [5]. AWS provides a scalable service, allowing for "on demand" running of code by increasing or decreasing usage depending on user requirements. However, more interaction with a service will cause a cost increase on the developer side.

1.2.2 Azure

Like AWS, Microsoft Azure is a cloud computing platform that comes with multiple services like Software as a Service (SaaS) and Platform as a Service (PaaS) [6]. Azure can be used to deploy different projects including machine learning models. While similar to AWS in nature, Azure is linked to mostly Microsoft services, while AWS provides a wider range of applications.

1.2.3 TorchServe

TorchServe [7] is a tool used for serving models built with PyTorch. The serivce provides compatibility, as the models for the project are all built with PyTorch. The service handles both HTTP and gRPC requests. It can serve multiple models simultaneously and can perform pre- and post-processing steps for data mutation.

1.2.4 Tensorflow Serving

Tensorflow Serving [8] is a similar service to TorchServe, but it is designed to work with models built using Tensorflow. Since the models for this project use PyTorch rather than Tensorflow, this solution would not be compatible.

1.3 Final Choice

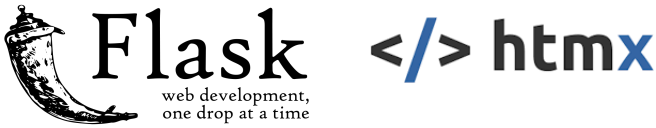

Figure 1: Services used for architecture

After considering all possible options, the most suitable option for a the project was to use Flask and HTMX. This is because Flask is the most lightweight option for small-scale projects, meaning it will have the most efficient performance overall. It is simplistic to set up and build, while providing good documentation for development of web applications. Flask also allows for future scalability should this project expand beyond the scope outlined within the brief.

Since the web application for the project is being deployed locally, the need for other paid or automated services mentioned above remains unnecessary. However, they may be considered in the future for expanding the application.

2 Implementation of Web Service

2.1 Architecture Overview

The web application is built on Flask [9], a lightweight and versatile Python web framework that is well-suited for small to medium web applications. Flask handles HTTP requests and routes them to appropriate Python functionality. It is often a popular choice for web services that requires flexibility with minimal overhead, ensuring clear and maintainable code.

2.2 Integrating HTMX

HTMX [10] was added to enhance the user experience by allowing interactions with the server without requiring full page reloads. This is particularly beneficial to display the data within the application dynamically, as the data loading is done asynchronously.

When a user inputs text for entity recognition, HTMX sends a request to Flask without disrupting the user's interaction with the page, updating the DOM with the processed results as soon as they are available.

2.3 Named Entity Recognition with SpaCy and DistilBERT

At the core of the application is the functionality for named entity recognition (NER) for abbreviations and long forms, powered by fine-tuned spaCy [11] and DistilBERT [12] models. SpaCy is a powerful and efficient library for natural language processing in Python, while DistilBERT is a lighter version of BERT that retains most of the model's accuracy but is more resource-efficient.

Both spaCy and DistilBERT have been fine-tuned for abbreviation and long-form recognition from their original NER parameters. Fine-tuning involves training a pre-trained model on a specific dataset that includes examples of the target entities. This process adapts the pre-trained models to be able to recognise and differentiate these entities to the context needs of the task.

2.4 Logging and Monitoring

To implement monitoring and debugging all activities and significant events, such as input, output, model used and timestamps, are stored within the "log_file.log". This is crucial to maintain the reliability of the application, providing insight into user interaction and any errors that might arise from unexpected inputs.

For page request errors, Flask has been configured with blueprints to give template page errors. In the future, it can be extended to include new error and custom messages within the request-response functionality.

A performance test was added to test for the speed of the application. This ensures adding new code changes can be reliably tested for speed and any issues can be resolved.

On the app, both performance and logs can be viewed from the admin panel login page which visualises the contents of both systems. Performance can be run with different options to test reliability and performance of the application.

2.5 Service

The implementation combines Flask's flexibility, HTMX's asynchronous capabilities and the ability to perform named entity recognition using spaCy and distilBERT. The setup not only optimises the user experience by providing quick and responsive web interactions but also ensures text processing is efficient by using the pre-trained models.

3 Performance of Web Service

This section talks about the performance of the web service created. It is assessed in four standards - size, speed, scalability, and accuracy.

3.1 Size

The deployed application hosts two models of different sizes. The spaCy model is the smaller model at 479MB, and DistilBERT is the larger at 760MB. The size of the model is important for managing the size of the website, as it is essential that the website would not become too large for cost effectiveness reasons. Thus, the spaCy model is more memory efficient than DistilBERT. However, after considering the sizes of both models, as the project deploys both models locally, resource limitations are passed to the machine running it.

3.2 Speed

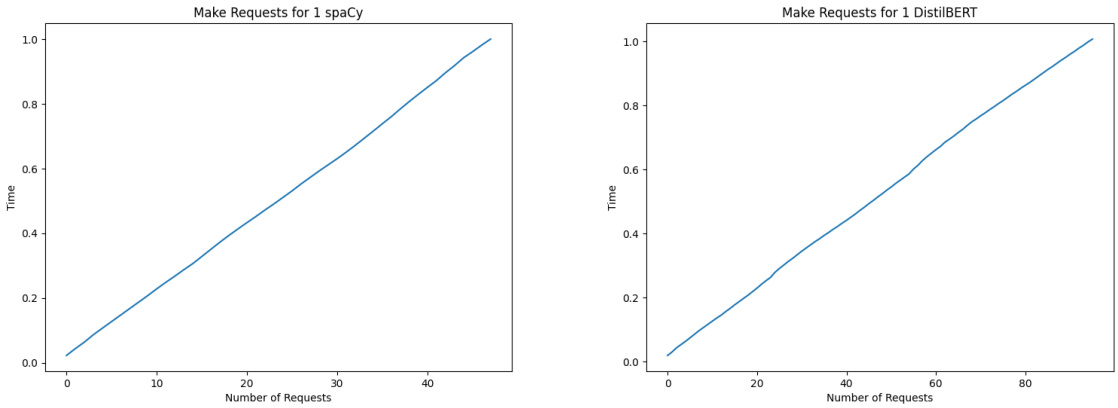

Figure 2: Requests made for one second using spaCy (left) and DistilBERT (right)

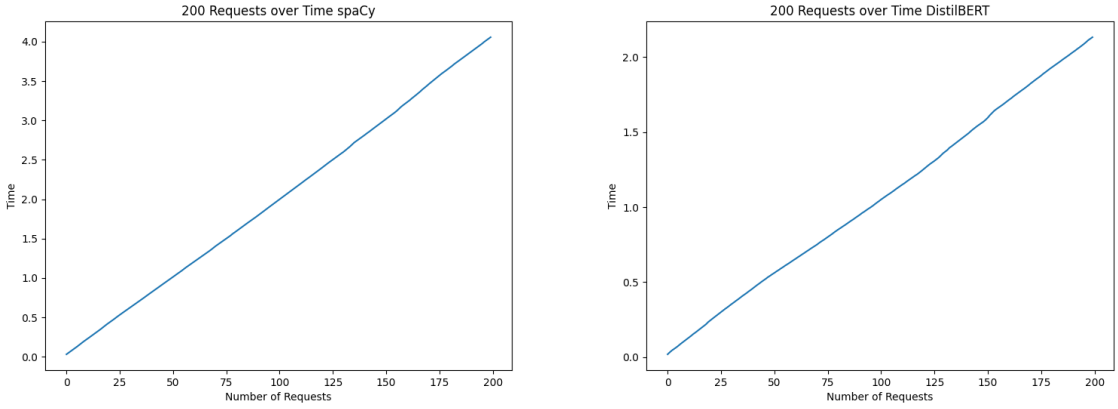

Figure 3: Accumulated time for 200 requests using spaCy (left) and DistilBERT (right)

The speed of the web service was evaluated in two ways. First method used was to accumulate the number of requests made per n second. The second method used for evaluation was to measure the time taken to make n requests. Both of these values could be changed for alternative tests. Figure (2) depicts the number of requests made for 1 second. SpaCy has generated over 40 requests per second, whereas DistilBERT has produced over 80 requests per second.

On the other hand, figure(3) represents how many seconds it took to make 200 requests. SpaCy took approximately 4 seconds to make 200 requests, while DistilBERT only took approximately 2 seconds to make 200 requests. For both test cases, these were made with short inputs.

In conclusion, these results show that DistilBERT can handle nearly double the number of requests than spaCy in a given time-frame.

3.3 Scalability

It is important to consider scalability when deploying a model online. As the user number increases, there may be updates to expand the site functionality. Moreover, with the increase in traffic, the site would need to be able to handle multiple requests concurrently, so that users can still get responses promptly. As our application is written using Flask, there is potential for scaling up the application by ensuring that the server uses threads or separate processes to handle multiple requests. Increasing the number of threads or processes will be more resource-intensive but will make the application more accessible to the users. Also, the application can be scaled up by having more servers to handle requests and increasing its storage resources to store a higher quantity of user data.

3.4 Accuracy

Figures (4) and (5) show the overall performance of the two models used within our web application in terms of how successful they are in producing correct predictions for acronym and long form detection. It can be seen that spaCy is the best performing model with higher overall F1 scores than DistilBERT. For example, spaCy's F1 score for predicting acronyms (AC labels) is 83% while for DistilBERT it is 80%. This means that using spaCy is most likely to give accurate responses. This could be explained by the longer training time for spaCy, as spaCy took 1104.01 seconds (not shown in the figure, taken from training data) to train, while DistilBERT only trained for 208.42 seconds.

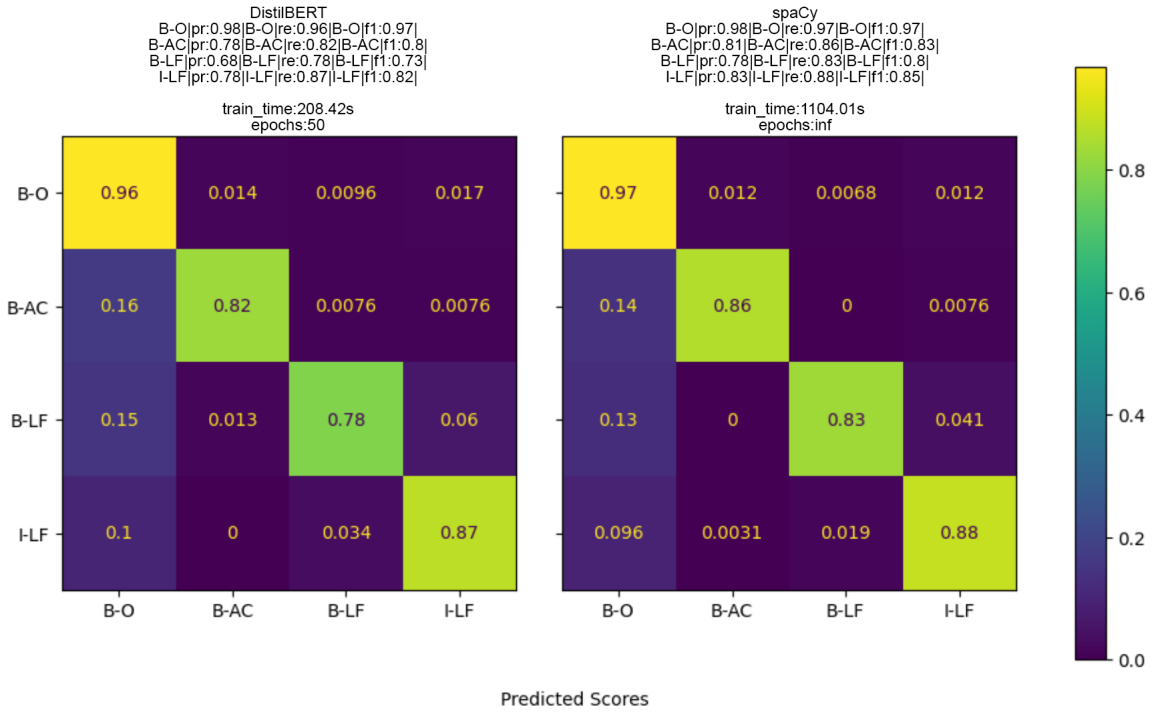

Figure 4: Confusion Matrix for each model

Experiment 1 Results		
Test	F1 Score	Time
DistilBERT	O:97%, AC:80%, BLF:73%, ILF:82%	208.42s
spaCy	O:97%, AC:83%, BLF:80%, ILF:85%	1104.01s

Figure 5: The Best Results of each model

4 Monitoring

4.1 File Monitoring

For monitoring user requests, we created a logging system that takes information about each request made and stores it into a text file (log_file.log). The stored information consists of the input, which is the text that the user enters, the prediction output by the model, the model type (spaCy or DistilBERT), and the time that the response was made. We store the time of the response so that we can track application usage and traffic in order to monitor the overall performance of the system. It is also useful for troubleshooting errors such as system crashes.

Table 1 shows example prediction results for sample text inputs for both models, demonstrating how the data is logged to the file.

Model Type	Input	Prediction	Time
spaCy	'OBS is an abbreviation'	[('OBS', 'AC'), ('is', 'O'), ('an', 'O'), ('abbreviation', 'O')]	'2024-05-18 02:27:48.243713'
DistilBERT	'OBS is an abbreviation'	[('ob', 'B-AC'), ('##s', 'B-AC'), ('is', 'B-O'), ('an', 'B-O'), ('abbreviation', 'B-O')]	'2024-05-18 02:27:51.087721'
DistilBERT	'Something different is not an abbreviation or a long-form!'	[('something', 'B-O'), ('different', 'B-O'), ('is', 'B-O'), ('not', 'B-O'), ('an', 'B-O'), ('abbreviation', 'B-O'), ('or', 'B-O'), ('a', 'B-O'), ('long', 'B-O'), ('##form', 'B-O'), ('!', 'B-O')]	'2024-05-18 02:28:21.994430'
spaCy	'Something different is not an abbreviation or a long-form!'	[('Something', 'O'), ('different', 'O'), ('is', 'O'), ('not', 'O'), ('an', 'O'), ('abbreviation', 'O'), ('or', 'O'), ('a', 'O'), ('longform', 'O'), ('!', 'O')]	'2024-05-18 02:33:39.567688'

Table 1: Example of data logged to the text file

4.2 Unit Tests

The unit tests conducted focus on input and output validation within the project, ensuring that further developments of the model do not break the previously established rules. The tests use the "run_model" function to perform the queries and test the inputs.

Test	**Expected Outcome**	**Actual Outcome**	**Status**
Test Input	type list[str]	type list[str]	Passed
Test Long Input	type list[str]	type list[str]	Passed
Test No Input	Empty list	Empty list	Passed

Table 2: Unit Tests

4.3 Functional Tests

Functional tests were performed to examine the behaviour of the app with user interaction. The tests outline if the app can perform its functional requirements. This includes the app opening and various interactions for named entity recognition.

Test	Expected Outcome	Actual Outcome	Status
Open application in a browser	Application opens in a web browser	Application opened in a web browser	Passed
Enter input into textbox	Text appears in the textbox	Text appeared in the textbox	Passed
Request tags via the generate tags button	Generate button will make a request	Generate button made a request	Passed
Response is received from server	Response returns from the server	Response returned from the server	Passed

Table 3: Functional Tests

5 CI/CD Pipeline

5.1 CI/CD

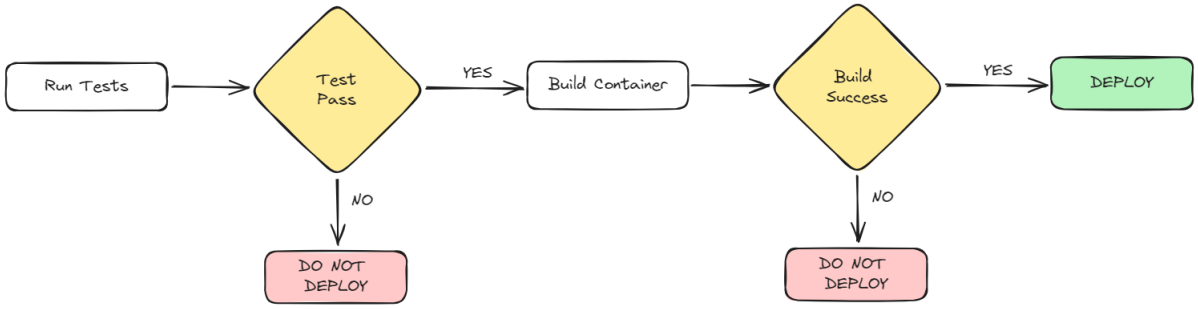

Figure 6: The Diagram of the Build Pipeline

The project currently employs Git with GitHub for source control management, ensuring that all code changes are tracked. In order to maintain the code quality and functionality, unit tests have been set up. The web service including the unit tests, runs in a Docker [13] container setup. PyTorch 2.2.1 with CUDA 12.1 image [14] is used to set up the Docker container. The container ensures a consistent environment across different machines, as well as compatibility to deploy onto external servers.

When code changes are made to the code base, the Docker container is rebuilt with the unit tests being re-executed to confirm correctness of code. If all tests pass, the Flask server is started within the Docker environment, indicating that the application is ready for local deployment. Robust in implementation, this manual setup is capable of handling the CI/CD pipeline locally.

In order to run the Docker container, the end user must ensure that they have Docker and WSL installed (if on windows). To build the container, within the app folder, the users must run *'docker build -t appname .'*. In order to run the container, the users must run *'docker run –rm -it -p 5000:5000 appname'*. The app name can be anything, while the run command designates the ports to run on locally.

5.2 Future Work

As part of future enhancements, a continuous integration tool that supports automation with GitHub should be included. Either AWS or Azure could be considered for paid tier upgrades.

For a more robust deployment process, integrating with a platform like Heroku would allow for a seamless pipeline. Heroku [15] already supports Docker-based deployments and can automatically push the application to a live environment once the tests pass. This would automate the current manual process of continuous deployment.

To improve scalability for multiple instances of the app, something like Kubernetes [16] could be used to run and serve multiple Docker instances at once.

References

[1] University of Surrey NLP Group. *Surrey-NLP/Plod-CW · datasets at hugging face*. (Accessed on 18/5/2024). 2024. URL: https://huggingface.co/datasets/surrey-nlp/PLOD-CW.

[2] M. Deery. *The Flask Web Framework: A Beginner's Guide*. Website. (Accessed on 18/5/2024). 2023. URL: https://careerfoundry.com/en/blog/web-development/what-is-flask/.

[3] detimo. *Python Flask: pros and cons*. Website. (Accessed on 18/5/2024). 2019. URL: https://dev.to/detimo/python-flask-pros-and-cons-1mlo.

[4] Eliza Taylor. *What Is Nginx? A Beginner's Guide*. Website. (Accessed on 18/5/2024). 2024. URL: https://www.theknowledgeacademy.com/blog/what-is-nginx/.

[5] AWS. *Amazon SageMaker*. Website. (Accessed on 19/5/2024). 2024. URL: https://aws.amazon.com/sagemaker/.

[6] Logan Mccoy. *Microsoft Azure Explained: What It Is and Why It Matters*. Website. (Accessed on 19/5/2024). 2024. URL: https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/.

[7] *TorchServe — PyTorch/Serve master documentation*. https://pytorch.org/serve/. (Accessed on 05/20/2024).

[8] *Serving Models - TFX - TensorFlow*. https://www.tensorflow.org/tfx/guide/serving. (Accessed on 05/20/2024).

[9] *Welcome to Flask — Flask Documentation (3.0.x)*. https://flask.palletsprojects.com/en/3.0.x/. (Accessed on 05/20/2024).

[10] *</> htmx - high power tools for html*. https://htmx.org/. (Accessed on 05/20/2024).

[11] *EntityRecognizer · spaCy API Documentation*. https://spacy.io/api/entityrecognizer. (Accessed on 05/20/2024).

[12] *DistilBERT*. https://huggingface.co/docs/transformers/en/model_doc/distilbert. (Accessed on 05/20/2024).

[13] *Install Docker Desktop on Windows | Docker Docs*. https://docs.docker.com/desktop/install/windows-install/. (Accessed on 05/20/2024).

[14] *Image Layer Details - pytorch/pytorch:2.2.1-cuda12.1-cudnn8-runtime | Docker Hub*. https://hub.docker.com/layers/pytorch/pytorch/2.2.1-cuda12.1-cudnn8-runtime/images/sha256-11691e035a3651d25a87116b4f6adc113a27a29d8f5a6a583f8569e0ee5ff897. (Accessed on 05/20/2024).

[15] *Cloud Application Platform | Heroku*. https://www.heroku.com/. (Accessed on 05/20/2024).

[16] *Kubernetes*. https://kubernetes.io/. (Accessed on 05/20/2024).