|  |  |
| --- | --- |
|  | Tech Immersion and Placement Programme  Applied Artificial Intelligence  6 January – 7 April 2020  Project Capstone Report  Submitted By:  Eugin Lee  (email@somewhere.com)  Koay Seng Tian  (sengtian@yahoo.com) |

Table of Contents

[1 Background 2](#_Toc36200718)

[1.1 Goal 2](#_Toc36200719)

[2 Methodology and Design 2](#_Toc36200720)

[2.1 Software Data Flow 2](#_Toc36200721)

[3 Applications 3](#_Toc36200722)

[4 Insights 3](#_Toc36200723)

[4.1 BERT Models Are Not Equal 3](#_Toc36200724)

[4.2 ROUGE and not BLEU 4](#_Toc36200725)

[5 Results and conclusions 5](#_Toc36200726)

[6 Recommendations 5](#_Toc36200727)

[7 Appendices 5](#_Toc36200728)

[7.1 GitHub 5](#_Toc36200729)

[7.2 Development Environment 6](#_Toc36200730)

[7.3 Extractive Summarizer 6](#_Toc36200731)

[7.4 Running the Summarizer 7](#_Toc36200732)

[8 References 8](#_Toc36200733)

# Background

This project is a collaboration between Nvidia and Republic Polytechnic (RP) as a capstone project for the Tech Immersion and Placement Programme (TIPP) programme. The project aims to provide a real- world work environment for the students to apply artificial intelligence techniques taught in the course. Supervisors in return receive the intellectual property of the final product.

## Goal

The goal of this project is to provide an annotation tool, to differentiate key takeaways in a corpus of text. The summarisation technique will be powered by Bidirectional Encoder Representations from Transformers (BERT) models.

# Methodology and Design

Extractive Summariser using BERT transformer model.

Using a webpage as an input, this BERT model summariser aims to extract key feature sentences of the main corpus.

These extracted features will then be presented as an annotated form, together with the main corpus as an output document.

Using a client-server model, the web application provides seamless transition between server (flask), user interface (Streamlit) and the underlying python code. The software is developed using open- source software, libraries and modules.

For evaluation, we will be using ROUGE scoring system which is optimised to calculate distances/ similarities between summarised articles. By using ROUGE, and comparing between BERT generated summaries against human generated summaries, we are able to calculate the similarity between the text and recommend ways to improve the results.

## Software Data Flow

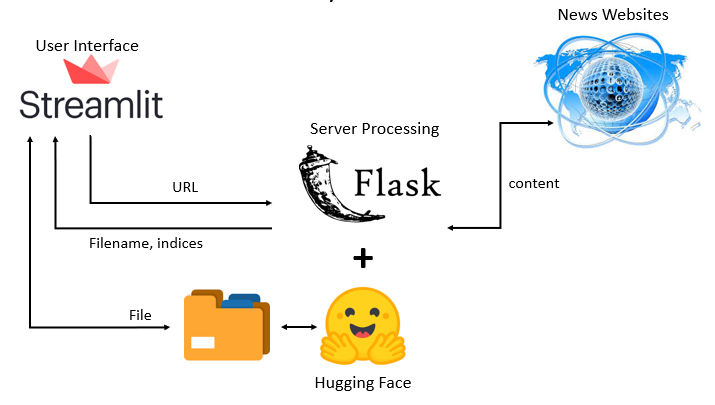


Figure 1 Software Data Flow

The software is developed using available open-source application frameworks (Streamlit, Flask) and Hugging Face BERT/Transformer model. The software is developed using Python scripting language.

Streamlit architecture is based on the ability to write web application the same way a plain Python script is written. Streamlit applications have a unique data flow. Any time something must be updated on the screen (for example, when the application need to response to a button is pressed), Streamlit will just rerun the entire Python script from top to bottom.

This will pose a challenge for the application/UI developer because it is not implemented as a 'callback', like most web applications will perform. Some of these quirks can be modified using Streamlit's cache decorator (streamlit@cache) which allows developers to skip certain costly computations when the application reruns. However, such modification, as we have observed, may create stability issues.

# Applications

In this project we utilised these libraries:

|  |  |  |
| --- | --- | --- |
| **Python Library** | **Usage** | **Installation** |
| Newspaper3k | Scraping tool | pip install newspaper |
| Pandas | Data manipulation | conda install pandas |
| summarizer | BERT summariser | pip install summarizer |
| datetime | Time stamp | - |
| Rouge | ROUGE scoring | pip install rouge |
| NLTK | Tokenising words | pip install NLTK |
| Flask | Server | pip install flask |
| Streamlit | User Interface | pip install streamlit |

# Insights

In this project we discovered two major takeaways that arise due to our experiments and research. We discovered that (1) model selection plays a big part of in operational efficiency, (2) and

## BERT Models Are Not Equal

When we first started out, we utilised the “stock” BERT model (bert-base-uncased) that was embedded in the summarizer library. While testing, we quickly realised that we may face into some usability issues as the waiting time for the extractive summary takes around 17 seconds to complete.

According to recent research on attention spans, current attention spans range between 8 to 12 seconds. This means that users might regard the seventeen seconds wait to be “too long” and may not even use the product even if it was a perfect product. We need to find a compromise between speed and performance.

To achieve this, we look at the different BERT models available in the market. We eliminated the “large” BERT models at first cut, because they contain more parameters which will mean longer processing time. This narrows our focus to the base BERT model. To evaluate the performance, we run a loop test, controlling all parameters except a model change. In this experiment, we recorded the performance in the form of processing time, and text length.

At this point, we are using just quantitative measures, such as (1) processing speed and (2) text length as our selection.

Within the BERT models, we chose DistillBERT as its processing speed falls within the sweet spot between 8-12 seconds and the summarized text length seems to be within an acceptable ballpark area. In addition, to artificially improve on the run time, we created a time illusion (to make time seem to pass faster), by inserting loading animation and stimulating an active program during the processing runtime.

|  |  |
| --- | --- |
|  |  |

Figure 2 Where You Got this from

## ROUGE and not BLEU

We selected ROUGE as our main technique to measure model accuracy. The main reason is because ROUGE provides more insights into the scoring mechanism. In ROUGE, the precision and recall scores are also provided which gives an understanding in the event of a low score, whether the machine generated text, is too short or too long, or is it not relevant. With this knowledge, we could either tweak the parameters correctly, or if the summary is poor, we can fine-tune it with a longer relevant corpus. With BLEU, another popular technique, only the final BLEU score (between 0 to 1) is given. This gives us no direction to improve.

ROUGE and BLEU are both popular methods in summaries evaluation. They use similar calculation methods surrounding n-grams and variations of recall and precision. When testing, we discovered that BLEU does not seem to work very well when there are differences in corpus length (between BERT and human generated). This could be due to the brevity penalty effect. The documentation recommended a smoothing technique to overcome this but we did not find the value to be as intuitively accurate as ROUGE.

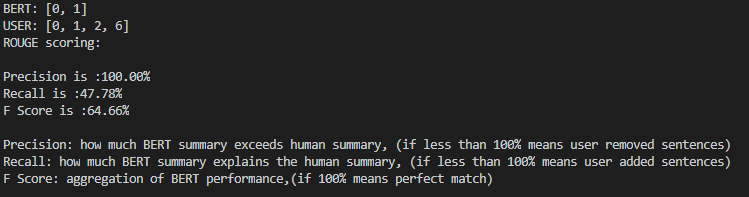


Figure 3 Please give a name

# Results and conclusions

We successfully created and deployed a web application that allows a user to (1) enter a URL and received a full text with annotated key points (2) user can provide feedback, which can be used to provide a quantitative scoring (3) we could improve the BERT model based on the scoring.

Our average runtime for the model is around 8-10 seconds. The model for some reason, seems to run slightly faster after a few iterations.

Project managers from Nvidia are extremely satisfied with the progress we made.

# Recommendations

Given that the code is designed to be modular based and robust, the engineer will be able to switch out part of the code further customised it. For example, as our parser is optimised for newspaper articles, it may not work as well for PDF documents. In which case, the engineer could switch in a PDF parser instead.

The Streamlit application works perfectly for a proof-of-concept user interface; to quickly unlock the power of the underlying model and test out concept feasibility. However, the UI content has some limitations such as feedback functionality like a HTML, and limited fine-tuning options, which hinders our ability to improve the product.

In Natural Language Processing (NLP) literature, the computer-generated summary is always compared against a “gold standard” summary to determine its accuracy. However, in our case, the BERT summary is compared against to a random user feedback. This user may have its own biasness and may have an inconsistent proficiency of the subject. This could affect our evaluation score of the model. To avoid this issue, it might be useful to collect an aggregation of user responses, before tweaking the model.

# Appendices

## GitHub

The source codes, documentation and the test data for the project are hosted in Github.

Repository URL: <https://github.com/koayst/rp_capstone>

Currently, the repository is set as private mode until the project is presented on 3rd April 2020. If you need to access it, please contact any of us.

Github uses Git as an open-source version control system. It helps is to keep the revisions straight, storing modification in a central repository. This allows us, as a developer, to easily collaborate, as we can download a new version of the software, make changes and upload the newest version, after we make the modifications.

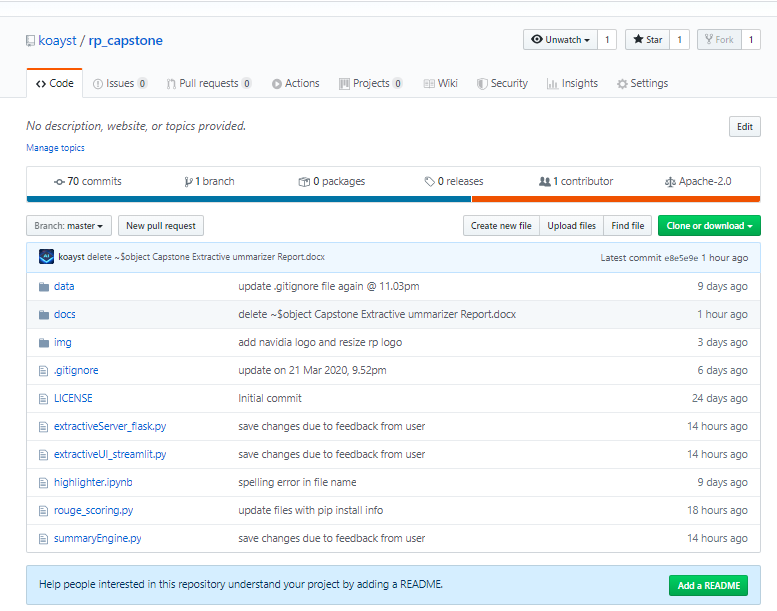


Figure 4 GitHub Repository Screen Shot

## Development Environment

We are using Windows Operating System (Windows 10 and Windows 8) as our development environment.

Anaconda Individual Edition (2019-10) was installed and then relevant modules are installed/updated as required by the project.

## Extractive Summarizer

* To run the system, it is advised to create a separate virtual environment. Use Git to bring down the source codes by cloning it from GitHub.
* Install the required libraries and modules as indicated in section 3.
* Open two terminals on a local machine (command prompt for Windows OS) i.e. one to run the Flask server and the other one to run the Streamlit webapp.
  + Flask server: *python extractiveServer\_flask.py*
    - Running on http://127.0.0.1:5500/ (Press CTRL+C to quit)
  + Streamlit: *streamlit run extractiveUI\_streamlit.py*
    - Local URL: <http://localhost:8501>
* Once Streamlit is running, you will notice the webapp is running in your browser. We are using Chrome browser.

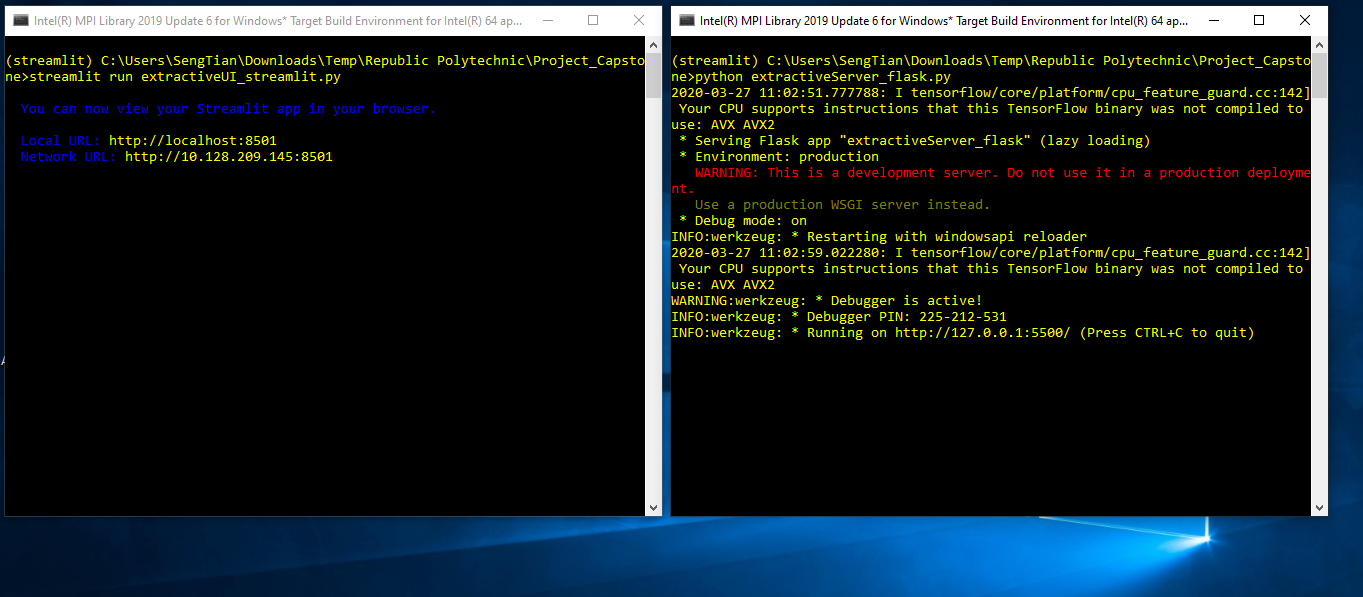


Figure 5 Screen shots for running the summarizer

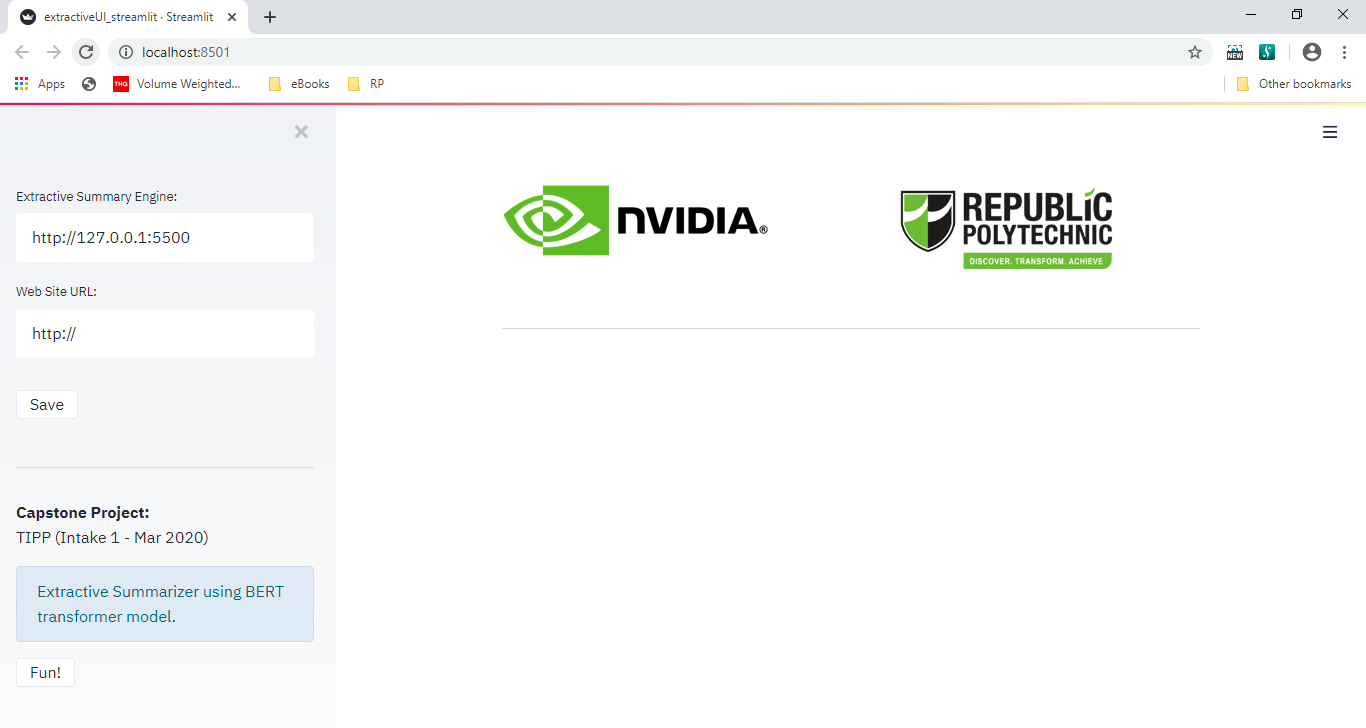


Figure 6 Streamlit webapp screen shot

## Running the Summarizer

* Go to Singapore’s Channel New Site (<https://www.channelnewsasia.com/>) and select a news article you want to summarize. Copy the URL and paste it to the Streamlit webapp. Press ‘Enter’ to start the summarizer engine.
* The output will be shown as a list of checkboxes. User can enhance the modelling by changing the checkboxes.
* User’s feedback can be saved by clicking the ‘Save’ button.

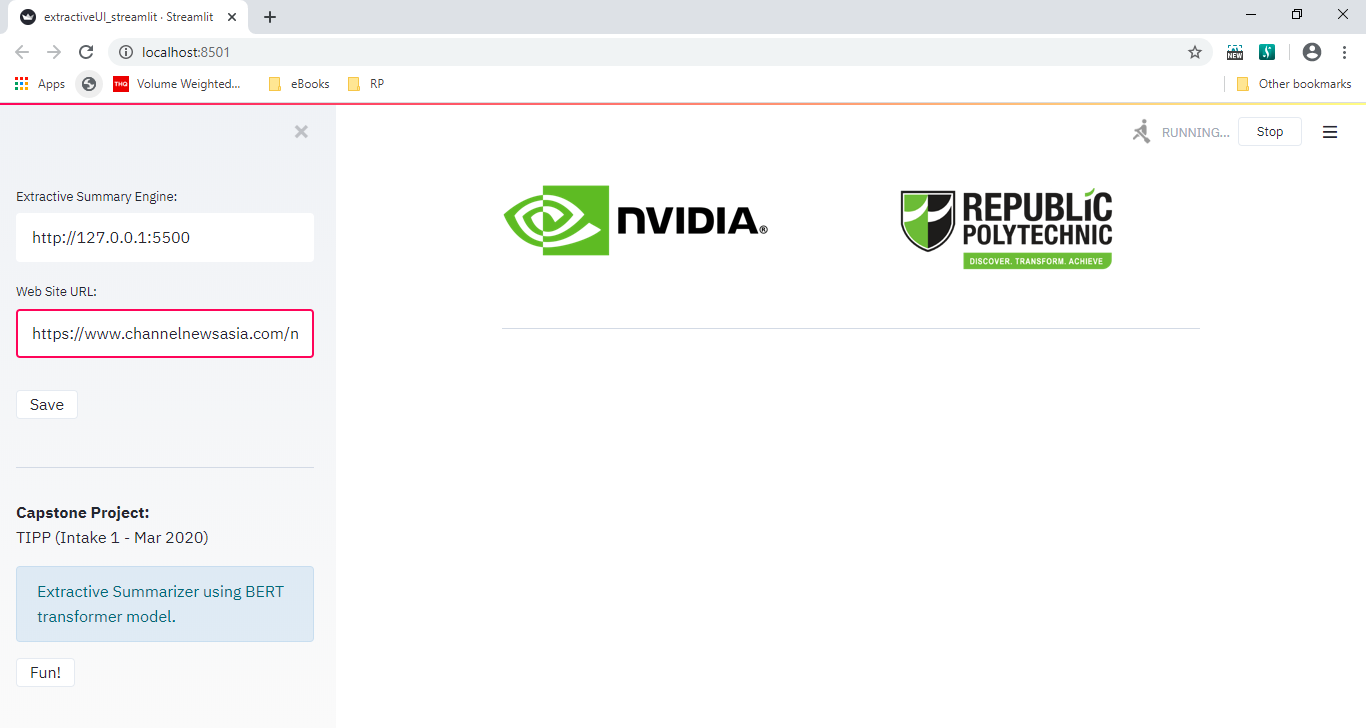


Figure 7 Input URL and Press ‘Enter’

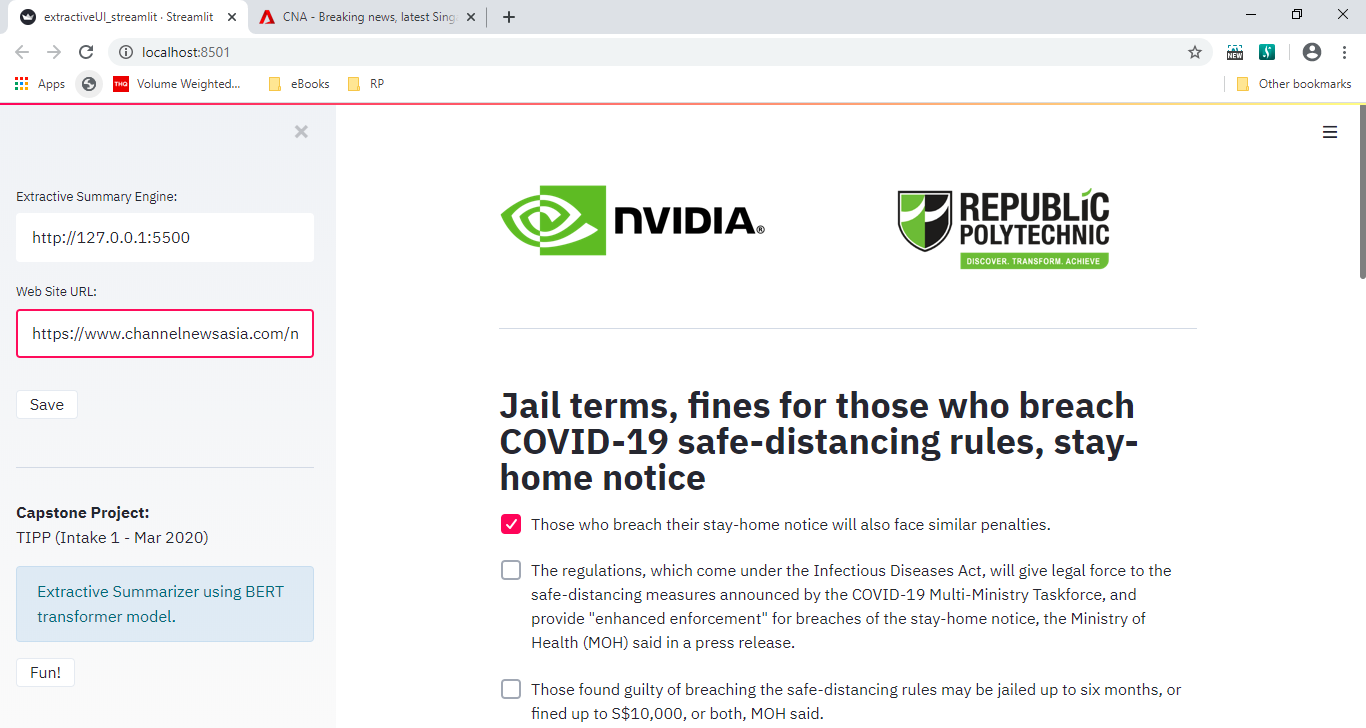


Figure 8 Output of Summarizer

# References

* Streamlit – <https://www.streamlit.io>
* Flask - <https://flask.palletsprojects.com>
* Eugin: Please ask your reference