Deploying Abstractive Text Summarizer using GCP

*Mohammed Zayd Jamadar
dept. Of Computer Science and Engineering
PES University
Bengaluru, India
zaydjamadar777@gmail.com

*Harshith Mohan Kumar
dept. Of Computer Science and Engineering
PES University
Bengaluru, India
harshithmohankumar@pesu.pes.edu

+Dr. S Natarajan
dept. Of Computer Science and Engineering
PES University
Bengaluru, India
natarajan@pes.edu

***Abstract*—Although emerging technology and a surplus of data are resulting in ground breaking research in the field of deep learning, deploying state of the art models into products is still a challenging task. In this work, we aim to expose the difficulties in deployment and operation to help familiarize individuals to the ML pipeline by utilizing various services on the Google Cloud Platform. We have implemented a state of the art transformer model, PEGASUS, in an end-to-end abstractive text summarizer product which has been deployed onto the internet for anyone to utilize.**

Keywords—machine learning, artificial intelligence, Google Cloud Platform (GCP), transformers, abstractive summarization, PEGASUS

I. Introduction

Over the past decade, deep learning models have shown incredible advancements. Each year a new architecture is proposed with extraordinary state of the art results. Machine learning (ML) models have shown to provide an enormous benefit in nearly all fields of business, ranging from marketing to scientific-,health- and security- related applications (Chen et al., 2012). An analytical study by Davenport (2006) describes companies which are able to leverage their data sources through analytical tools tend to achieve a substantial competitive advantage. However, unlike regular software applications machine learning models require an unprecedented amount of compute power. This constraint severely limits the application of these state of the art models in consumer electronics.

The wide-spread application of ML is rather young and therefore still confronted with many obstacles (Rudin et al., 2014). Published state of the art ML research is not driving sufficient real-world impact due to the performance differences between real-world data and the confided test/train datasets. The entire deployment pipeline is a crucial area which requires careful technical implementation. We will take a closer look at the ML pipeline which we implemented for our custom model in the V. DEPLOYMENT section.

Also during this time, Natural Language Processing (NLP) has experienced an incredible amount of growth including a significant increase of deep-learning based models which can map an input sequence into another output sequence. These sequence-to-sequence models have been a popular solution for problems such as machine translation (Bahdanau et al., 2014), speech recognition (Bahdanau et al., 2015) and video captioning (Venugopalan et al., 2015). However, text summarization is a very different problem. Summarization is the process of minimizing the length of the source text while keeping the integrity of the message. This process often does not depend very much on the length of the source unlike the other problems we looked at earlier. In addition, there are two different categories of summarization: Extractive summarization and Abstractive summarization.

Extractive Summarization extracts or copies parts from the original input text based on the sources computed using either statistical features or linguistic features. The machine should be able to understand the complete text and be able to pick the important phrases. Abstractive Summarization is the task of generating a short and concise summary that captures the important ideas/features of the input text. These generated summaries usually regularly contain new sequences of phrases and sentences which would not have been observed in the source text. Therefore the machine should be able to not only comprehend the sequences of words but also be able to formulate relevant new sequences of sentences. Due to the real-word knowledge and semantic class analysis, abstractive summarization is generally a harder task than extractive summarization. It is also better since the summary is an approximate representation of a human generated summary, making it more meaningful.

II. Related Work

Significant advancements approached NLP problems when a team at google brain discovered the transformer model. Prior to the discovery of this attention based model, Recurrent Neural Networks (RNN) were utilized to perform sequential computation. Soon an attention mechanism was utilized to improve the performance of the RNN by teaching the decoder to pay attention to the hidden states of the decoder. However the fundamental constraint of sequential computation remained. Inorder to address this problem the google brain team removed the recurrence and instead relied entirely on an attention mechanism to draw global dependencies between input and output. This Transformer model allowed for significantly more parallelization and was able to reach a new state of the art shortly after a few hours of training. In recent years, variations of the Transformer model have resulted in extraordinary models such as BERT and GPT.

In this product we implemented a specific variation of the Transformer model by google known as PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. PEGASUS showed remarkable performance with large datasets, and didn't require a large number of examples for fine-tuning to get near state-of-the-art performance.

III. Overview

The whole project can be divided into two sections namely the Frontend React Website and the Backend Model Inference Pipeline. The React Website is hosted on the Firebase Hosting Service provided by Google. The pipeline is located in a Flask App framework.

We deploy the Pegasus model as a stateless microservice. When a client makes a request, the incoming request contains all the necessary information for the API to fulfill it. The API does not need to record the information from the previous requests.We can easily add or remove new instances of the application, on demand, whenever necessary.

IV. Deployment

When building an ML model the level of complexity of the model can vary drastically on a case-to-case basis. However, the underlying framework is more or less the same. Typically, data coming in from the source is followed by a series of data cleaning and preprocessing straps and then we extract or create important features from the input data for the model to train on. Once the model is trained it can be exposed to unseen data for making predictions through some sort of API. This last stage is what creates a handful of challenges. Fig. 2 visually demonstrates the connection and flow between the different stages of the ML model pipeline.

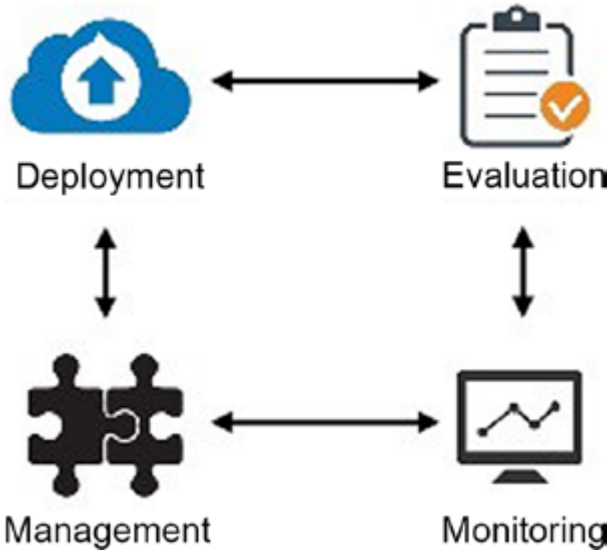

Fig. 1. Project Development Cycle

To understand the deployment process behind our product we will be taking a look behind the front end and the backend technologies which piece together the product. We decided early on that we would be using the google cloud platform for all of our cloud services and for hosting we decided to use firebase for its ease of access and compatibility with google cloud platform.

A. Firebase

To start things off we hosted our website using firebase web hosting. Firebase hosting is a production grade web content hosting platform for developers. It's popularly used to quickly deploy web apps and serve both dynamic and static content to a global content delivery network (CDN). Firebase hosting is free. It also by default provides SSL certificate and offers an impressive speed across multiple locations. These factors ultimately led us to choose firebase as our hosting service.We built and deployed a simple and minimalist website using the React JavaScript library.

B. Cloud Run

A majority of the work on the backend is handled by cloud run. In short, cloud run is a fully managed infrastructure that scales from zero instantly. It allows developers to develop apps and deploy them easier and faster than ever before. Another wonderful thing about cloud run is the fact that we pay as much as we use. During inactive requests, the container will turn off and for since the server is not utilizing any resources we aren't charged. This allows us to minimize our cost as much as possible. We developed a flask app which would receive a GET request from the firebase web application. It would then utilize the pretrained stored model within the container itself to compute the necessary summary of the provided input text. Finally, it returns a JSON response back to the firebase application.

C. Cloud Storage Bucket

Since a majority of the work is done by the cloud run container, there's not much on the google storage bucket. However, we did utilize this tool to keep all of the versions of our container. Everytime we built a container we would store the image as a bucket. This allowed us to roll different versions and test very easily.

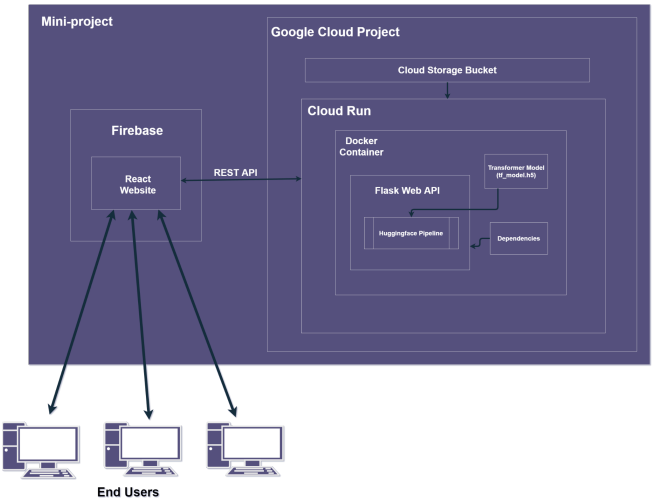

Fig. 2. System Design

V. Methodology

For the Web API we created a bare-bone Flask App which has the Huggingface pipeline in the form of a Flask method. The method accepts the URL, parses it and gets the text. This is later passed on to the pipeline for inference. It then returns the result i.e the abstractive summary. The Pegasus model [7] is located in the same directory and is used by the Huggingface pipeline.

The Flask app along with the Model and config file is packaged using Docker into a container image. A requirements file is necessary to be placed inside the project directory for this. Packaging was done using a script. The image is then uploaded on the Cloud Storage Bucket of GCP Project.

Cloud Run uses the docker container image located in the Cloud Storage Bucket. Before that it has to be configured with the number of CPU's, memory and region information.

The docker container is run on the Cloud Run . It automatically generates a URL to access the API. The Cloud Run API is based on REST-ful architecture. It can be called with a HTTP GET request.

Fig. 3 Tensorboard for the Custom Model

VI. Training

We also trained a attention based transforme model from scratch based on [6]. The model was created using Tensorflow's functional api. Fig. 3 shows the train and test loss of the model. Later on, we found out that Tensorflow does not have any methods to save custom sub-classed models. The only solution was to serialize the model layer by layer and write a custom save method on our own. Since this method was cumbersome and out of scope of the project, we decided to deploy an out of the shelf model from the Google-research Repository.

VII. Conclusion

At the end we built a fully functionally end-to-end ML product utilizing one of the state of the art models for anyone to use over the internet. This product is capable of summarizing any piece of input text using abstractive summarization techniques. All of the tools required to deploy and host our website is provided by the google cloud platform. We used firebase hosting for our website and paired it up with cloud run to run our pretrained PEGASUS model.

VIII. Future Work

During this entire process we aimed to focus on trying to maximize the compute power while minimizing the cost of the project. I believe this severely undercuts the performance gains. In the future we would like to optimize our code to be more memory efficient. This would help solve the expensive memory allocation issue and also improve the backend request latency since the container will be more lightweight. Experimenting with other cloud platforms would also be a great idea to compare the differences and determine which flows better. If we had the time we would have also put more thought into the UI design for our website.

IX. Reference

[1] Chen, H., R. Chiang, and V. Storey (2012). "Business Intelligence and Analytics: From Big Data to Big Impact." Mis Quarterly 36 (4), 1165–1188

[2] Davenport, T. H. (2006). "Competing on analytics." Harvard business review 84 (1), 98–107

[3] Baier, Lucas et al. "Challenges in the Deployment and Operation of Machine Learning in Practice." ECIS (2019).

[4] Dzmitry Bahdanau, Kyunghyun Cho, & Yoshua Bengio. (2016). Neural Machine Translation by Jointly Learning to Align and Translate.

[5] Subhashini Venugopalan, Marcus Rohrbach, Jeff Donahue, Raymond Mooney, Trevor Darrell, & Kate Saenko. (2015). Sequence to Sequence – Video to Text.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, & Illia Polosukhin. (2017). Attention Is All You Need

[7] Jingqing Zhang, Yao Zhao, Mohammad Saleh, & Peter J. Liu. (2019). PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization.