Evaluation engine for School exams

DISSERTATION

Submitted in partial fulfillment of the requirements of the

MTech Data Science and Engineering Degree programme

By

Yashodeep Vaidya

2019HC04042

Under the supervision of

Gaurav Desai, Director

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Pilani (Rajasthan) INDIA

Feb, 2022

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the Dissertation entitled Evaluation engine for School exams and submitted by Mr. Yashodeep Vaidya ID No. 2019HC04042 in partial fulfillment of the requirements of DSECLZG628T Dissertation, embodies the work done by him/her under my supervision.

Signature of the Supervisor

Name: Gaurav Desai

Designation: Director

Place: Mumbai

Date: 27-Feb-2022

Abstract

In online school exams especially theory exams where students write long essays like textual answers it is difficult to quickly and quantitatively score them. Manual process is time consuming and error prone. In this paper, we are trying to create a system for automatic and accurate evaluation of such textual answers using Abstractive Text summarization and text similarity checking algorithms. Teacher should provide a model answer for each question while setting the exam. Students will submit answers as images and this system will take these images as inputs and process them to estimate score.

Keywords. NLP, Text Summarization, Abstractive Text Summarization, Text similarity

Acknowledgements

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of this **M.Tech Data science and Engineering** project. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work.

I wish to thank my supervisor **Mr. GAURAV DESAI** for his valuable and constructive suggestions during the planning and development of this project work. His willingness to give his time so generously has been very much appreciated.

I would also like to extend my thanks and gratitude to my additional examiner **Ms Sankara nayaki K** for providing me an opportunity to carry this project, along with purposeful guidance and moral support extended to me throughout the duration of the project work.

VAIDYA YASHODEEP PRASAD PRIYANKA

Table of Contents

Table of Contents

Abstract 3
Acknowledgements 4
Table of Contents 5
Introduction 6
  Process Flow 6
  Benefit of the solution 6
Architecture 7
  Basics of Text Summarization 7
  Extractive Text Summarization 8
  Abstractive Text Summarization 8
  Basics of Text Similarity Detection 9
    BERT with cosine distance 10
  Brief description of framework 12
    Hugging Face Transformers [1] 12
    React JS[3] 21
  Solution Architecture 22
Implementation 25
  Text Summarization Flow 25
    Extractive Summarization 26
    Abstractive Summarization 27
  Text Similarity using Cosine similarity 28
    Code Snippet 28
Summary 29
Conclusion and Recommendations 30
Future Scope 31
Bibliography 32
Reference 33
Appendix 34
Checklist of items for the Final report 35

Introduction

This project aims to implement an evaluation engine for estimating marks to be awarded to the students for their textual answers. It will use techniques in Natural Language Processing and Deep learning to estimate the correctness of the given answer. It aims at making evaluation of textual answers more quantitative, transperant and automated. In the existing setup evaluation of text answers and essays is mostly subjective and it is left to the judgement of the evaluator.

Process Flow

1. Input model answers in the system
2. Input student answers in the system
3. Text Summarization creates a summary of both model answers and student's answers.
4. Compute Text similarity of both the summaries.
5. Based on percentage similarities, score the answer. If similarity is 0.9% then score the answer 90% of total marks.

Benefit of the solution

This is useful to any ed-tech platform where exams / tests are conducted. This product will be an add-on to those platforms and will aid in faster evaluations of answers and quicker test results.

Architecture

Basics of Text Summarization

Text summarization refers to the technique of shortening long pieces of text. The intention is to create a coherent and fluent summary having only the main points outlined in the document. Automatic text summarization is a common problem in machine learning and natural language processing.

Skyhoshi, who is a US based machine learning expert with 13 years of experience and currently teaches people his skills, says that “the technique has proved to be critical in quickly and accurately summarizing voluminous texts, something which could be expensive and time consuming if done without machines. “

Machine learning models are usually trained to understand documents and distill the useful information before outputting the required summarized texts.

Main approaches to automatic summarization

1. Extractive Text summarization
2. Abstractive Text summarization

Extractive Text Summarization

The Extractive text summarization technique involves pulling keyphrases from the source document and combining them to make a summary. The extraction is made according to the defined metric without making any changes to the texts.

Example:

Source - Joe and Mary rode on a donkey to attend the annual event in Mumbai. In the city, Mary gave birth to a child named Jesus.

Extractive summary: Joe and Mary attend event Mumbai. Mary birth Jesus.

Abstractive Text Summarization

The abstractive technique entails paraphrasing and shortening parts of the source document. When abstraction is applied for text summarization in deep learning problems, it can overcome the grammar inconsistencies of the extractive method. The abstractive text summarization algorithms create new phrases and sentences that relay the most useful information from the original text just like humans do. Therefore, abstraction performs better than extraction. However, the text summarization algorithms required to do abstraction are more difficult to develop; that’s why the use of extraction is still popular.

Example: *Joe and Mary came to Mumbai where Jesus was born.*

In this project we will be using Abstractive Text summarization as it is more semantically and grammatically correct.

Basics of Text Similarity Detection

Text similarity has to determine how ‘close’ two pieces of text are both in surface closeness [lexical similarity] and meaning [semantic similarity]. For instance, how similar are the phrases “the cate ate the mouse” with “the mouse ate the cat food” by just looking at the words?

- On the surface, if you consider only word level similarity, these two phrases appear very similar as 3 of the 4 unique words are an exact overlap. It typically does not take into account the actual meaning behind words or the entire phrase in context
- Instead of doing a word for word comparison, we also need to pay attention to context in order to capture more of the semantics. To consider semantic similarity we need to focus on phrase/paragraph levels where a piece of text is broken into a relevant group of related words prior to computing similarity. We know that while the words significantly overlap, these two phrases actually have different meanings.

Text similarity detection method we are using in this paper is BERT with cosine distance.

BERT with cosine distance

Bidirectional Encoder Representations from Transformers (BERT)

The Language modeling tools such as ELMO, GPT2 and BERT allow for obtaining word vectors that morph knowing their place and surroundings.

Our goal here is to show that the BERT word vectors morph themselves based on context. Take the following three sentences for example

- record the ***play***
- ***play*** the record
- ***play*** the game

The word *play* in the second sentence should be more similar to *play* in the third sentence and less similar to *play* in the first. We can come up with any number of triplets like the above to test how well BERT embeddings do.

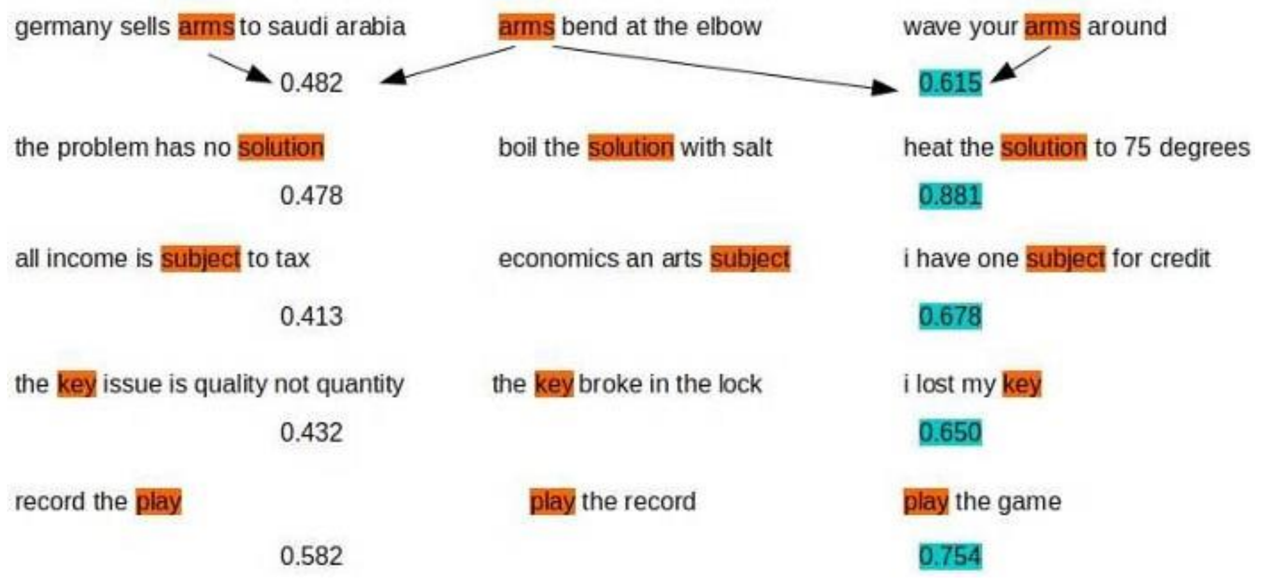
germany sells arms to saudi arabia
arms bend at the elbow
wave your arms around
0.482
0.615
the problem has no solution
boil the solution with salt
heat the solution to 75 degrees
0.478
0.881
all income is subject to tax
economics an arts subject
i have one subject for credit
0.413
0.678
the key issue is quality not quantity
the key broke in the lock
i lost my key
0.432
0.650
record the play
play the record
play the game
0.582
0.754

Fig - 1

BERT embeddings are contextual. Each row shows three sentences. The sentence in the middle expresses the same context as the sentence on its right, but different from the one on its left. All three sentences in the row have a word in common. The numbers show the computed cosine-similarity between the indicated word pairs. BERT embedding for the word in the middle is more similar to the same word on the right than the one on the left.

Brief description of framework

Hugging Face Transformers [1]

Transformers provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio.

These models can applied on:

- Text, for tasks like text classification, information extraction, question answering, summarization, translation, text generation, in over 100 languages.
- Images, for tasks like image classification, object detection, and segmen tation.
- Audio, for tasks like speech recognition and audio classification.

Transformer models can also perform tasks on **several modalities combined**, such as table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.

Transformers provides APIs to quickly download and use those pretrained models on a given text, fine-tune them on your own datasets and then share them with the community on our model hub. At the same time, each python module defining an architecture is fully standalone and can be modified to enable quick research experiments.

Transformers is backed by the three most popular deep learning libraries —Jax, PyTorch and TensorFlow — with a seamless integration between them. It's straightforward to train your models with one before loading them for inference with the other.

Features

- Easy-to-use state-of-the-art models:
    - High performance on natural language understanding & generation, computer vision, and audio tasks.
    - Low barrier to entry for educators and practitioners.
    - Few user-facing abstractions with just three classes to learn.
    - A unified API for using all our pretrained models.
- Lower compute costs, smaller carbon footprint:
    - Researchers can share trained models instead of always retraining.
    - Practitioners can reduce compute time and production costs.
    - Dozens of architectures with over 20,000 pretrained models, some in more than 100 languages.
- Choose the right framework for every part of a model's lifetime:
    - Train state-of-the-art models in 3 lines of code.
    - Move a single model between TF2.0/PyTorch/JAX frameworks at will.
    - Seamlessly pick the right framework for training, evaluation and production.

- Easily customize a model or an example to your needs:
    - We provide examples for each architecture to reproduce the results published by its original authors.
    - Model internals are exposed as consistently as possible.
    - Model files can be used independently of the library for quick experiments.

Reasons for selecting Hugging Faces Transformers

Following are the advantages of Hugging face transformers

1. Ease of use
2. Large selection of models and constantly new models are getting added in the library
3. Ability to fine tune models using tokenizers
4. Ability to train models on new data

Transformers architecture[2]

The Transformer in NLP is a novel architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies with ease. The Transformer was proposed in the paper Attention Is All You Need. It is recommended reading for anyone interested in NLP.

Transformer's Model Architecture

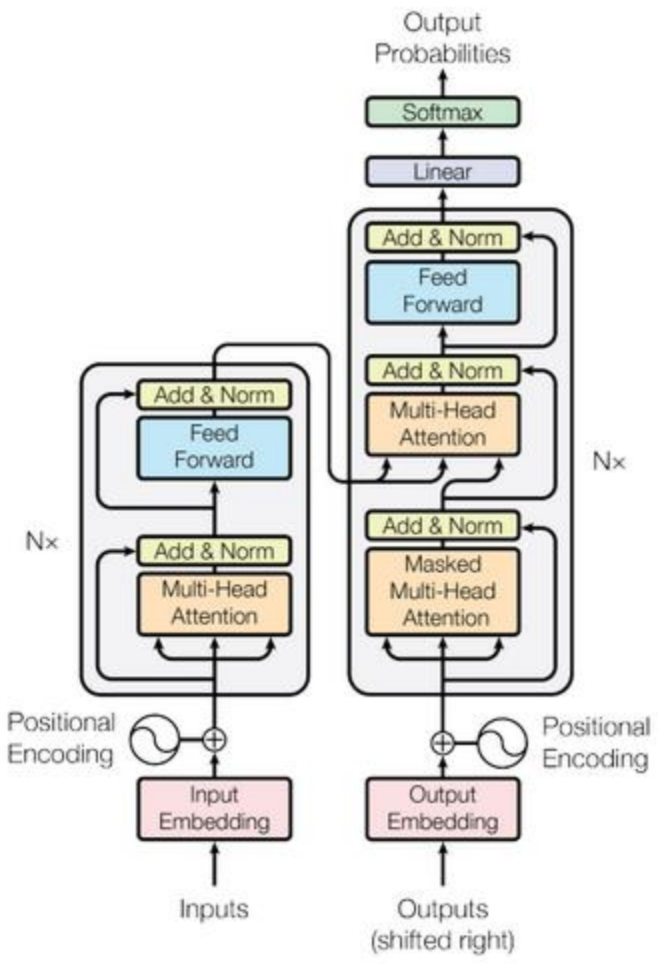

Fig - 2

Now focus on the below image. The Encoder block has 1 layer of a Multi-Head Attention followed by another layer of Feed Forward Neural Network. The decoder, on the other hand, has an extra Masked Multi-Head Attention.

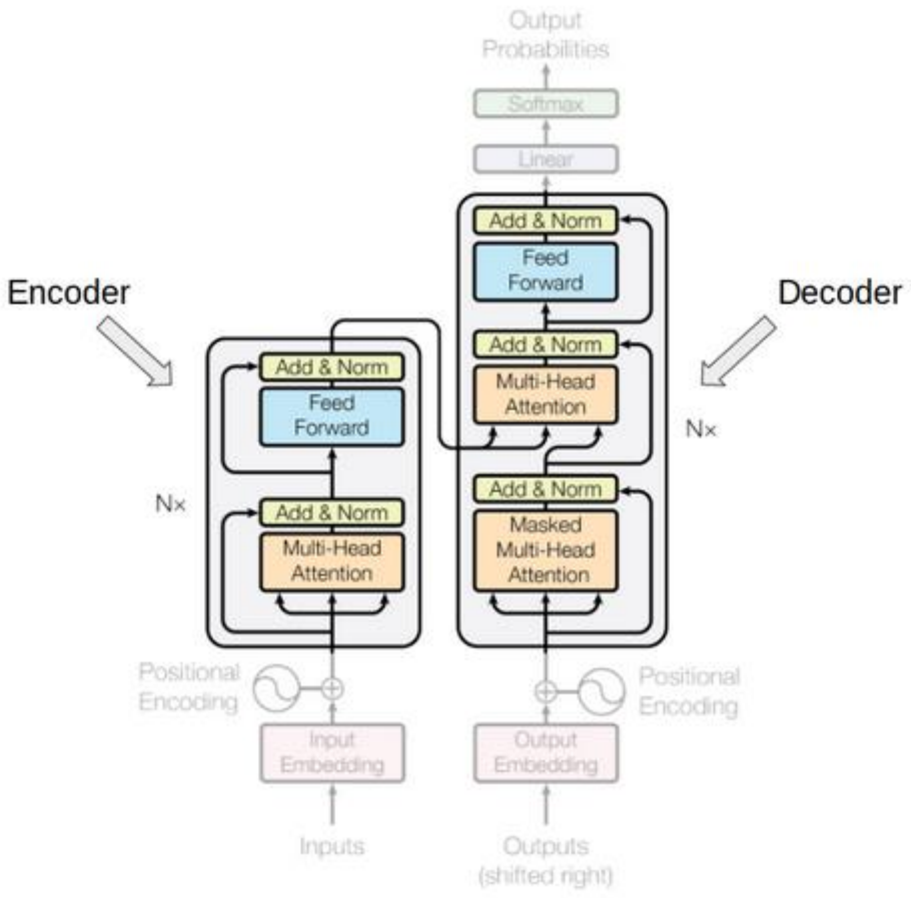

Fig - 3

The encoder and decoder blocks are actually multiple identical encoders and decoders stacked on top of each other. Both the encoder stack and the decoder stack have the same number of units.

The number of encoder and decoder units is a hyperparameter. In the paper, 6 encoders and decoders have been used.

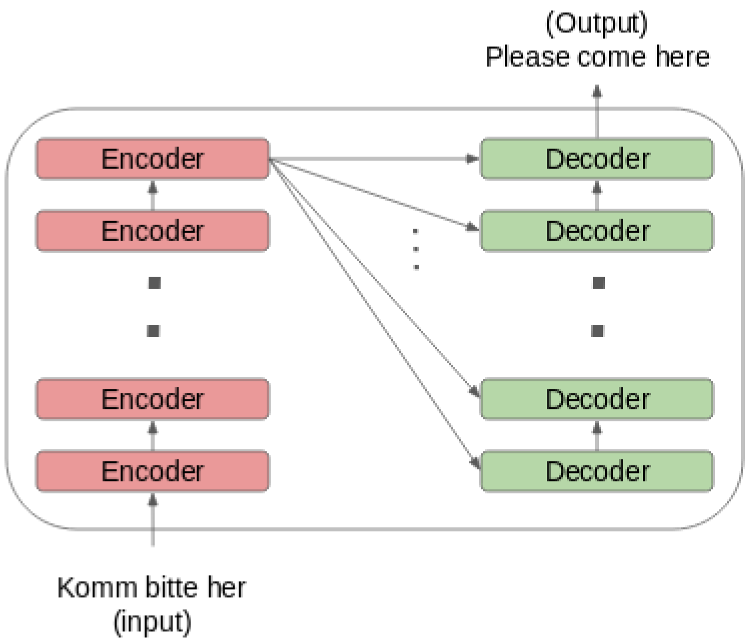

Fig - 4

Let’s see how this setup of the encoder and the decoder stack works:

- The word embeddings of the input sequence are passed to the first encoder
- These are then transformed and propagated to the next encoder
- The output from the last encoder in the encoder-stack is passed to all the decoders in the decoder-stack as shown in the figure below:

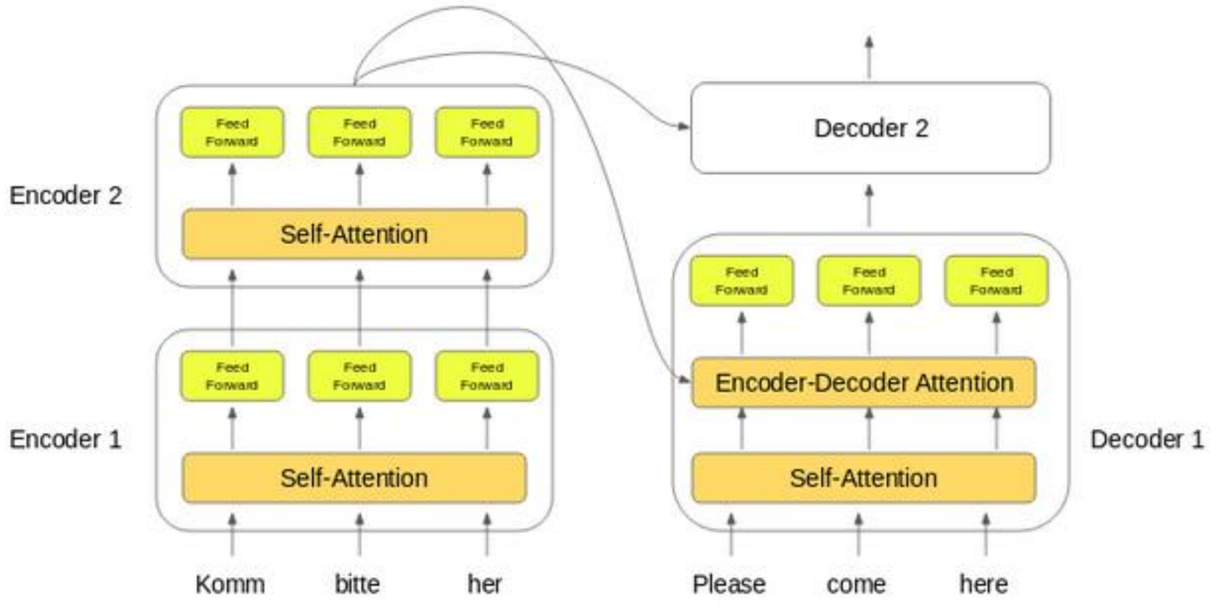

Fig - 5

An important thing to note here – in addition to the self-attention and feed-forward layers, the decoders also have one more layer of Encoder-Decoder Attention layer. This helps the decoder focus on the appropriate parts of the input sequence.

You might be thinking – what exactly does this "Self-Attention" layer do in the Transformer? Excellent question! This is arguably the most crucial component in the entire setup so let's understand this concept.

Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks. Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the extensions to add such functionality to the application. Some of the popular Flask extensions are discussed later in the tutorial.

Features of Flask

Below are some of the features of flask.

- built-in development server and fast debugger
- integrated support for unit testing
- RESTful request dispatching
- Jinja2 templating
- support for secure cookies (client side sessions)
- WSGI 1.0 compliant
- Unicode based

Reason of selection of Flask framework in project

In this project UI is developed using flask, as it is open source and have below advantages.

- Flask has a lightweight and modular design, so it easy to transform it to the web framework you need with a few extensions without weighing it down
- ORM-agnostic: you can plug in your favourite ORM e.g. SQLAlchemy
- Basic foundation API is nicely shaped and coherent.
- Flask documentation is comprehensive, full of examples and well structured. You can even try out some sample application to really get a feel of Flask.
- It is super easy to deploy Flask in production (Flask is 100% WSGI 1.0 compliant")
- HTTP request handling functionality
- High Flexibility: The configuration is even more flexible than that of Django, giving you plenty of solutions for every production need.

React JS[3]

React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding.

React Features

- JSX − JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
- Components − React is all about components. You need to think of everything as a component. This will help you maintain the code when working on larger scale projects.
- Unidirectional data flow and Flux − React implements one-way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional.
- License − React is licensed under the Facebook Inc. Documentation is licensed under CC BY 4.0.

Solution Architecture

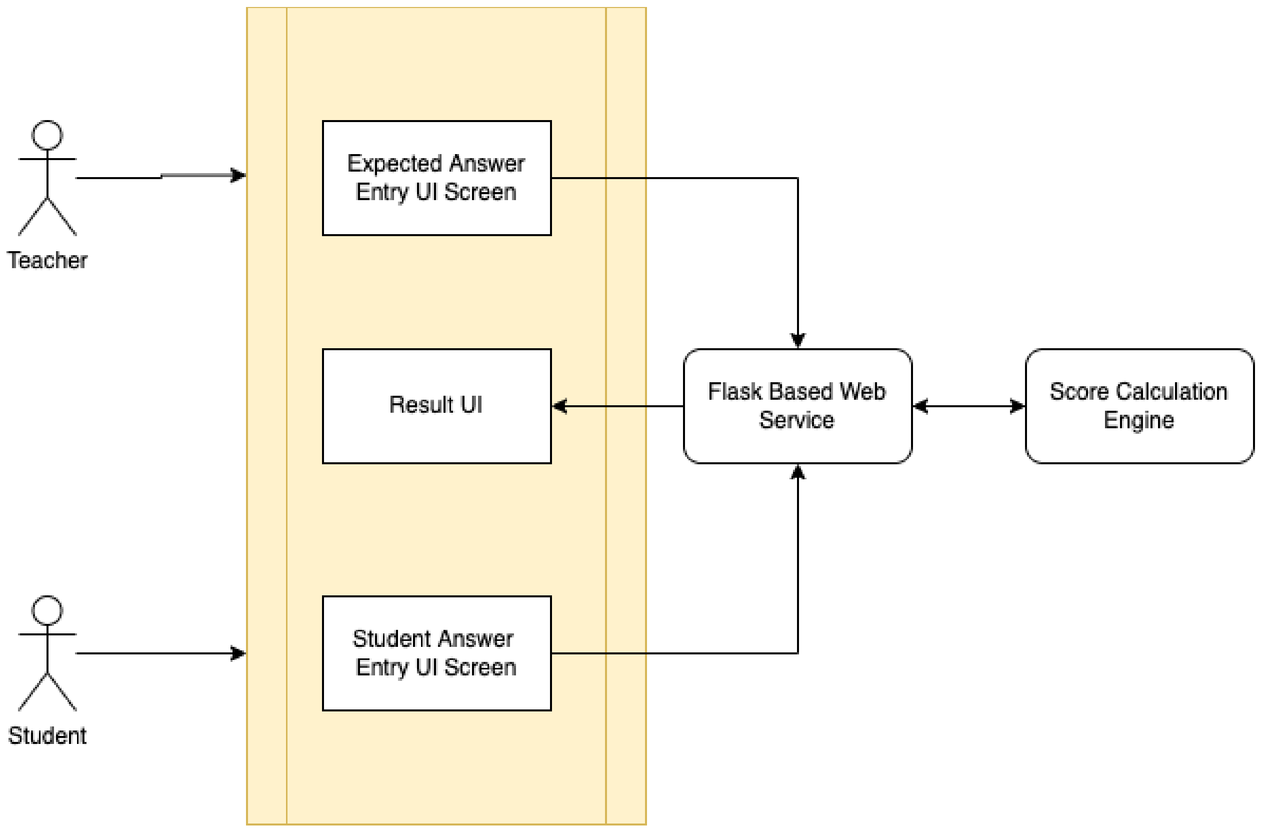

Fig - 6

- As shown in the above architecture diagram, we are providing a User interface where teacher can enter question as well as expected answer in the UI screen
- In the second screen student will be presented with the question and a text area to enter actual answer
- Both expected and actual answers are sent to Flask Based web service
- Flask web service endpoint will call evaluation engine which will call summarizer and similarity evaluation procedures.
- Similarity evaluation process will calculate score

Implementation

Text Summarization Flow

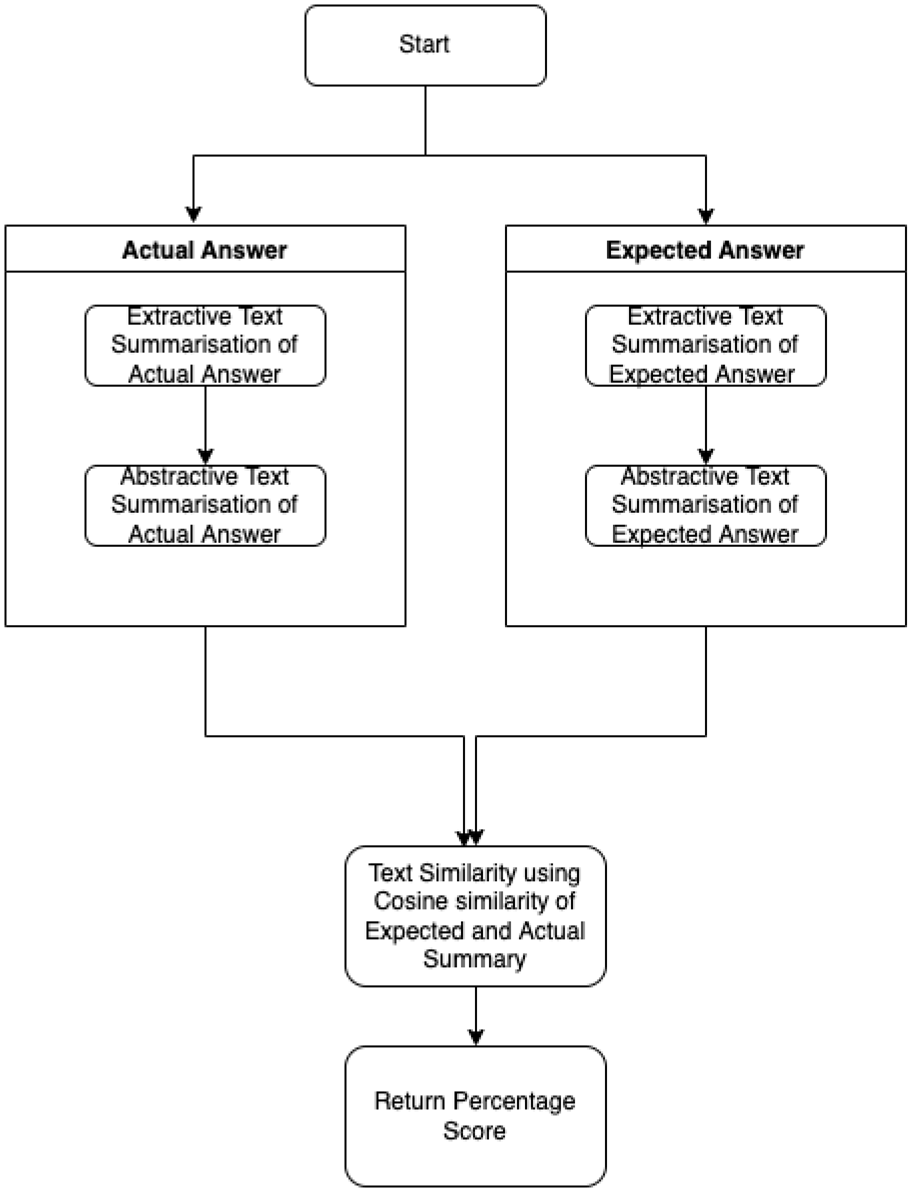
Start
Actual Answer
Extractive Text Summarisation of Actual Answer
Abstractive Text Summarisation of Actual Answer
Expected Answer
Extractive Text Summarisation of Expected Answer
Abstractive Text Summarisation of Expected Answer
Text Similarity using Cosine similarity of Expected and Actual Summary
Return Percentage Score

Fig - 7

Hugging face models for summarization have a limitation that the max length of text supported by the models is very small as compared to actual answers or essays to be summarized hence to improve performance we have first used Extractive text summarization using LSA summarizer and output of this is given to Abstractive model to generate a summary on that.

Extractive Summarization

LSA Summarizer

Latent semantic analysis is an unsupervised method of summarization it combines term frequency techniques with singular value decomposition to summarize texts. It is one of the most recent suggested technique for summarization

Code snippet

```
# Set up Extractive Summarizer

from sumy.parsers.plaintext import PlaintextParser

from sumy.nlp.tokenizers import Tokenizer

import nltk

nltk.download('punkt')

from sumy.summarizers.lsa import LsaSummarizer as Summarizer

from sumy.nlp.stemmers import Stemmer

from sumy.utils import get_stop_words
```

```python
from sumy.summarizers.lsa import LsaSummarizer

summarizer_extractive = LsaSummarizer()


def extractive_summary(t, c):
 parser_extractive = PlaintextParser.from_string(t, Tokenizer("english"))
 summary = summarizer_extractive(parser_extractive.document, c)
 sumText = ""
 for sentence in summary:
  sumText += str(sentence)
 return sumText
```

Fig - 8

Abstractive Summarization

Using Hugging face transformer based model for summarization

Code snippet

```python
# Set up Abstractive Summarizer
from transformers import pipeline
model = "facebook/bart-large-cnn"


from transformers import AutoTokenizer
tokenizer_abstractive = AutoTokenizer.from_pretrained(model)
summarizer_abstractive = pipeline("summarization", model=model, tokenizer=tokenizer_abstractive)
```

```
def abstractive_summary(t):

 return summarizer_abstractive(t)
```

Fig - 9

Text Similarity using Cosine similarity

Code Snippet

```
from sentence_transformers import SentenceTransformer, util

sentences = [exp_sum, act_sum]


model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')


#Compute embedding for both lists

embedding_1= model.encode(sentences[0], convert_to_tensor=True)

embedding_2 = model.encode(sentences[1], convert_to_tensor=True)


util.pytorch_cos_sim(embedding_1, embedding_2)
```

Fig - 10

Summary

#	Task	Expected date of completion	Names of Deliverables	Status
1	Low level analysis	Jan -22	Flowcharts / Documents	Completed
2	Data Preparation	Jan -22	Documents / Python Code	Completed
3	Text Summarization	Jan – 22	Python Code	Completed
4	Text Similarity Detector	Feb - 22	Python Code	Completed
5	Representation either as Notebook or using ReactJS based UI	Mar - 22	Notenbook / UI Code	Completed

Conclusion and Recommendations

The Ed-Tech ecosystem is growing very rapidly and Natural Language based solutions like the one shown in this document will definitely help automate the manual work involved in evaluation of textual answers.

Future Scope

Improve accuracy of Text Summarization

Although in this project we have used Extractive and then Abstractive summarization to summarize large essays but accuracy of it is still not very high, need more research on how we can improve Text summarization efficiency for large texts.

OCR

Optical character recognition is not implemented due to time constraints but we can add this feature in the app to allow students to directly upload answers in image format and OCR will extract actual answers from them and will use it in the pipeline.

Bibliography

[1] - Hugging face documentation https://huggingface.co/docs/transformers/index

[2] - Analytics Vidya https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/

[3] - Tutorials point - https://www.tutorialspoint.com/reactjs/reactjs_overview.htm

Reference

Natural Language Processing with Python by Edward Loper, Ewan Klein, and Steven Bird

Hugging face models

https://huggingface.co/models?pipeline_tag=summarization&sort=downloads

Appendix

1. BERT - Bidirectional Encoder Representations from Transformers
2. NLP - Natural Language Processing
3. OCR - Optical Character Recognition

Checklist of items for the Final report

Is the Cover page in proper format?	Y
Is the Title page in proper format?	Y
Is the Certificate from the Supervisor in proper format? Has it been signed?	Y
Is Abstract included in the Report? Is it properly written?	Y
Does the Table of Contents page include chapter page numbers?	Y
Does the Report contain a summary of the literature survey?	Y
Are the Pages numbered properly?	Y
Are the Figures numbered properly?	Y
Are the Tables numbered properly?	Y
Are the Captions for the Figures and Tables proper?	Y
Are the Appendices numbered?	Y
Does the Report have Conclusion / Recommendations of the work?	Y
Are References/Bibliography given in the Report?	Y
Have the References been cited in the Report?	Y
Is the citation of References / Bibliography in proper format?	Y