End-to-End QA Generation Using Pre-Trained Transformer Models

Ryan Lee and Mahalakshmi Balasubramanian

April 2023

Presentation: Click Here
Code: Click Here

1 Introduction

Natural Language Processing has exploded in popularity and applicability over the years. With the advent of chatbots like ChatGPT the general public has become more keen and aware of these technologies. But these language models are still fairly limited in what they can and more work is required before they achieve intelligent behavior. Due to this limitation proper benchmarks are needed to evaluate the effectiveness of any language model. One of the most popular benchmarks is answer generation. Given any question the goal is to generate the most probable answer that satisfies the question. In our project, we explore a less popular domain which is question and answer generation. Given a context generate question and answer pairs that satisfy the given context. We use pre-trained transformers model to achieve this goal and the simplicity of our project is that no training or fine-tuning is needed. This means more accessibility because GPUs are not needed to do inference. Additionally, we also discuss future applications of our project in the context of education.

2 Question and Answer Generation in Detail

First we go over the the four variants of question and answer generation then we go into a specific example. The first three variants are referenced from [1]. Extractive QA tries to extract the answer from the context. Open Generative QA, generates any question and answer based on the context. Closed Generative QA is not given any context and it is allowed to generate anything. Abstractive QA focuses on generating answers to more open-ended questions [2]. Our project uses a combination of Extractive and Generative QA Variants. Answer generation is Extractive whereas Question creation is Generative. Now we go over a specific example of generating QA pairs. Suppose the context is "Atoms are the building blocks that come together through chemical bonding to form

molecules in the universe. In this model of a molecule, the atoms of carbon (black), hydrogen (white), nitrogen (blue), oxygen (red), and sulfur (yellow) are in proportional atomic size. The silver rods indicate chemical bonds that hold the atoms together in a specific three-dimensional shape." Then an example of a QA pair is "What are atoms?" and "building blocks."

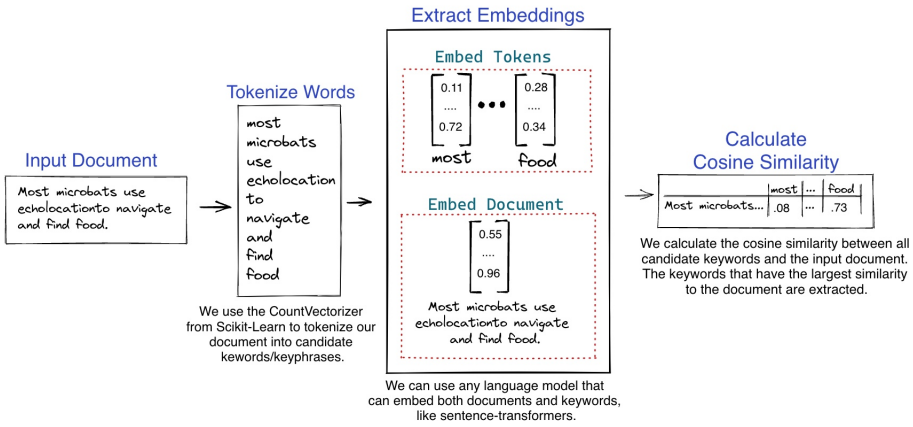
3 What is New or Different With This Implementation?

This implementation introduces several new and different things. The first is that we explore end-to-end Question and Answer generation which is not a well-explored area. There are models which can predict an answer given the context and question and few other models generate question given the context and answer. But this implementation generates multiple question answer pairs given the context. It only requires context as the input. The second is that we use the latest hugging face transformer models like T5 for end-to-end generation based on the logic outlined in [3]. This paper will serve as our main source of inspiration for this project. We also attempt to generate multiple question and answer pairs for the given context. Most implementations are concerned with just generating one pair. We also tried to apply this implementation to an educational setting. For example, we tried to come up with assessment questions for a chapter in Principles of Biology Textbook[12].

4 Models

Here we discuss all four models used in our implementation. We also discuss the Haystack QA framework which was used to preprocess the data we used. **Haystack** is an open-source framework used for building search systems that work intelligently over large document collections [4]. First we use the TextConverter method which extracts text from files and returns it in the document format to be used [5]. Then we use the PreProcessor method to format the text to our specifications [6]. PreProcessor cleans any whitespaces, blank lines and splits the document in to contexts which is then used as input to the keyword extraction models.

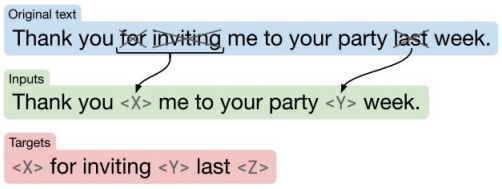
KeyBERT is a method used for keyword extraction that leverages BERT embeddings and simple cosine similarity to create keywords and key phrases in a document that are the most similar to the document itself. The following is a concise diagram from [7] that explains how KeyBert works.



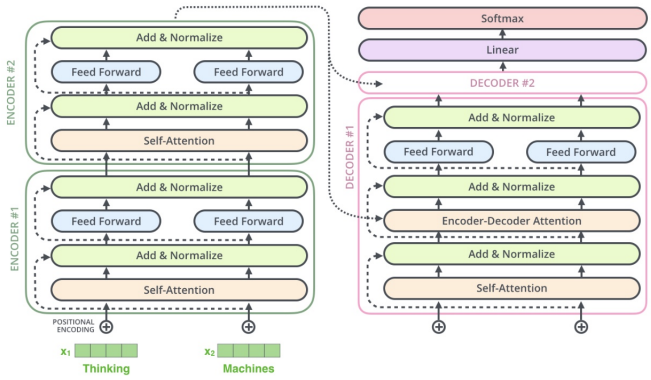
First, given a text or document as input, KeyBERT uses Countvectorizer to tokenize words and extract candidate keywords and key phrases. Then it uses a pre-trained BERT model to embed both the documents and keywords. In the last step, it calculates the cosine similarity between all candidate keywords and the input document and finally selects the words which has the highest similarity with the given document.

Another keyword extractor used in this implementation is **Yake** which stands for **Y**et **A**nother **K**eyword **E**xtractor and will be referenced from [8]. Unlike other keyword extractors, Yake uses an unsupervised approach rather than being trained against any corpora. The details will be left out and referenced but at a very high level, Yake uses six steps in the following order: Text preprocessing, feature extraction, individual terms score, candidate keywords list generation, data deduplication, and ranking.

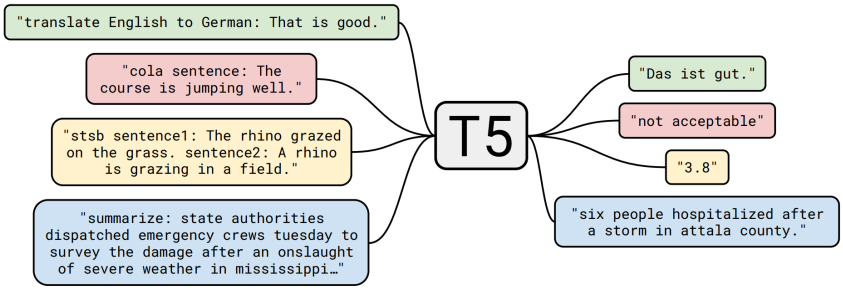
T5 is a text-to-text transfer transformer model where the input and output are text strings. it is an encoder-decoder model and converts all NLP problems into a text-to-text format. It is trained using teacher forcing. This means that for training, we always need an input sequence and a corresponding target sequence. T5 was introduced in this paper [9].T5 was first pre-trained on the C4 (Colossal Clean Crawled Corpus) data-set which is approximately 700 GB in size. Then it is fine tuned on various text-to-text tasks such as language translation and fill in the blanks. Below is an example that it was fine tuned on from the paper.



We also included a diagram showing the architecture from [10].



We can see that the underlying architecture is almost identical to any standard transformer. Below is a diagram from the reference that shows examples of T5 usage.

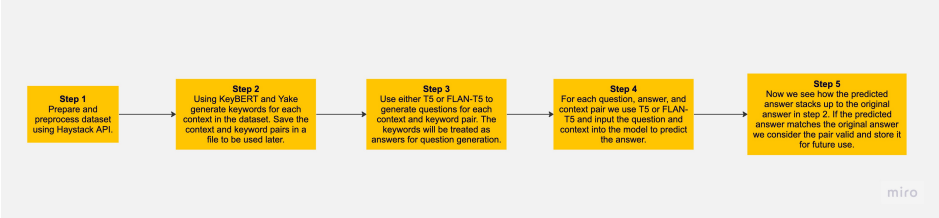


Next up is FLAN-T5 which is almost identical to T5. FLAN-T5 even has the same number of parameters, the only difference is that the model has been fine-tuned on more than 1000 additional tasks covering also more languages [11]. We would be considering this model in future for further improsising the implementation.

5 Implementation

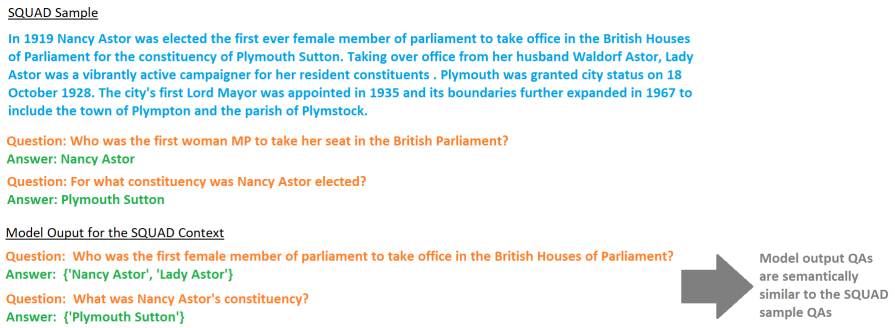
Our implementation consists of five steps in total. First we preprocess the dataset using the Haystack API. We use this biology book [12] and only preprocess Chapter 2. Once the preprocessing is done, we use two keyword extractors KeyBERT and Yake to extract keywords for each context and store them for later use. Next, we use T5 model finetuned on QG(Question Generation) to generate questions for the keywords and the context. The keywords are treated as pseudo answers. At the end of this step, we have a file where each row contains questions, answers, and the given context. Next we use T5 model finetuned on QA(Question Answering) to generate an answer for each question. We see how the predicted answer matches the original keyword created near the beginning

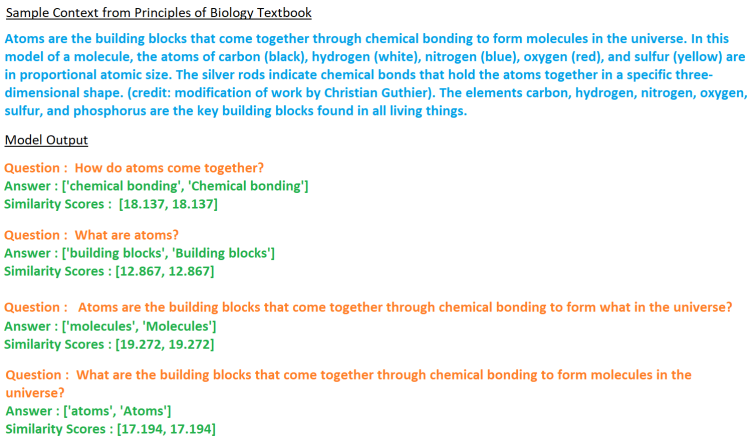
and if the two semantically match up we consider them valid and store it. For checking semantic similarity between the original keyword and the predicted answer and between the context and the answers, we use similarity scores calculated using the sentence transformer models. So far, for most of the steps we have stuck to T5. Specifically in step 3 we use this model [13] and in step 4 we use this model [14]. In step 5, we use this model [16] to check for answers that semantically match. Below we have provided a flowchart that summarizes this implementation.



6 Results

Using the implementation logic explained above, we were able to generate approximately 80 questions for the second chapter from Principles of biology textbook. Apart from this, we also tried to validate the implementation with some of the sample contexts from SQUAD dataset that already has a question and answer pair. The question and answer generated through our implementation semantically matches with the already available question answer pair in the SQUAD dataset. Also, we were able to generate multiple question answer pairs for the chosen sample context from the SQUAD dataset. Below Figure illustrates this.





As seen from the above figure, our model was able to generate question answer pairs for a sample content extracted from a biology textbook. However, there are two parameters in the model that can be modified to filter out undesired QA outputs. One is the keyword_weight and the other is the match_threshold. In Step 2, Yake and KeyBERT assigns a weight to each extracted keyword from the context. Our model lets the user choose a value for keyword_weight to filter out keywords that do not satisfy a specified weight. In the final step, the model uses Semantic Answer Similarity (SAS) to compare the relevance of the answers (keyword and the predicted answer) to the question. SAS model assigns a similarity score to each answer and if the score between the answers do not exceed match_threshold, it considers that QA pair as valid.

7 Future Improvements

Our current model works well however we have ideas that can be used to improve its output quality in the future. We mainly used different implementations of T5. We are yet to use FLAN-T5 and in the future we would like to experiment with this model. We want to see how FLAN-T5 compares to T5 for the steps listed in the implementation. We would also like to fine-tune FLAN-T5 specifically for end-to-end question answer generation and see how it compares to fine tuned T5. Our current implementation does not do any training or fine-tuning which is something we are interested in for future use. Introducing parameters that could further fine tune the quality of Question answer pairs generated is also something that we could explore. Additionally, our implementation generates multiple QA pairs that can be used in education domain create assessments, however in the future we want to create a model that can generate an exhaustive list of abstractive and extractive QA pairs. This would be a huge aid to any student studying because they would have a long list of new questions to work on. We also have a plan to host this implementation in Hugging face space along with an inference API.

References

[1] https://huggingface.co/tasks/question-answering

[2] https://docs.pinecone.io/docs/abstractive-question-answering

[3] Lopez, Luis and Cruz, Diane and Cruz, Jan and Cheng, Charibeth (2020) *Transformer-based End-to-End Question Generation*

[4] https://haystack.deepset.ai/overview/intro

[5] https://docs.haystack.deepset.ai/docs/file_converters

[6] https://docs.haystack.deepset.ai/docs/preprocessor

[7] https://maartengr.github.io/KeyBERT/api/keybert.html#keybert._model.KeyBERT

[8] https://medium.com/@adityamishra.rishu/keyword-extractor-yake-35870de21a0d

[9] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu (2020) *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*

[10] http://jalammar.github.io/illustrated-transformer/

[11] https://huggingface.co/google/flan-t5-xxl

[12] Elizabeth O'Grady, Jason Cashmore, Marsha Hay, Carol Wismer (2017) *Principles of Biology - An Introduction to Biological Concepts*

[13] https://huggingface.co/mrm8488/t5-base-finetuned-question-generation-ap

[14] https://huggingface.co/MaRiOrOsSi/t5-base-finetuned-question-answering

[15] https://huggingface.co/docs/transformers/model_doc/t5

[16] https://huggingface.co/sentence-transformers/multi-qa-distilbert-dot-v1

[17] Semantic Answer Similarity for Evaluating Question Answering Models Julian Risch, Timo Möller, Julian Gutsch, Malte Pietsch