# UAQuestion Answering System: UnforBERTable!

Computer Science Department
University of Fort Smith Arkansas
Noah Buchanan, Joshua Buster, Alana Matheny

July 2021

**Abstract**

In this paper we present our research of Question answering systems here at the University of Arkansas-Fort Smith. The BERT language model is the key driver behind our system, this combined with document retrieval from a tokenized corpus of files, and the fine-tuning of BERT for question answering on the SQuAD dataset is the topic of our research.

## Introduction

Our system UnforBERTable is a system designed to answer questions in an open domain format. The model makes use of Inverted Indexes, RTF-IDF, Cosine Similarity, and a fine-tuned BERT model trained for the specific task of question answering on the SQuAD dataset (Stanford Question Answering Dataset). BERT (Bidirectional Encoder Representations from Transformers) is still a fairly new state of the art language model, it first made its debut in 2018 and was published by Jacob Devlin and his colleagues from Google[4]. BERT was the first language model to jointly condition on both left and right context in all layers[4]. BERT boasts an impressive 88.5 F1 Score on the SQuAD 1.1 dataset when it first debuted, which was our initial inspiration to try and utilize it for question answering as they did. Our goal was to attempt to achieve results that were at least in a decent margin of the original results with our own trained model on SQuAD 2.0, the distinction between 1.1 and 2.0 being that once trained on 2.0 BERT should also be able to make the distinction for unanswerable questions.

## Background

Closed domain question answering systems have been around for a while and are not considered a relevant or popular cutting edge approach to question answering anymore. In their research paper "Answer Complex Questions: Path Ranker Is All You Need."[2] Huawei says that,

> "Currently, the most popular method for open-domain Question Answering (QA) adopts "Retriever and Reader" pipeline, where the retriever extracts a list of candidate documents from a large set of documents followed by a ranker to rank the most relevant documents and the reader extracts answer from the candidates."

Before we can rank our corpus documents we will first have to accomplish the task of training our model to understand context and language this is where BERT comes in. In there paper Devlin et al. define BERT as follows.

> Bidirectional Encoder Representations from Transformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task specific architecture modifications.

BERT is our key to a effeciently trained model with a high F1 score to boot. With this key to our models training in terms of context and language we are ready to fine tune our model to achieve the desired result of performing the task of question answering.

## Specification

The technique for our system functionality first begins with a corpus of files that will be used for finding relevant documents. We utilized JavaCC tokenization to tokenize a large corpus of files and save these to an output file. Once tokenized it is time to build the Inverted Index. An Inverted Index is essentially a way to map terms to specific documents in a lightning fast fashion relative to the sheer size of documents. There are two parts, dict.raf and post.raf, .raf indicating random access files so there is no need for linearly reading every line to get to what we need. Inside dict.raf it will look essentially like a disk optimized hash table, dict.raf will contain terms, the postings number, and the amount of documents that term was in. It is placed inside dict.raf like a hash table, we take the hashcode of the term mod m, m being the total size of dict.raf. We used a load factor of 2 so there are twice as many spaces as there are unique terms in dict.raf, each empty space is denoted with a -1. Linear probing is used to insert and extract results. post.raf is filled in such a way while creating these two files that if you take the posting number of a term and its document count and go to the line of the posting number in post.raf and go down the exact doc count, you will have looked at all files that contain the specific term where we obtained these values. This is the basic concept behind our document retrieval. Each postings entry has an RTF-IDF value.

Relative Term Frequency(RTF) and Inverse Document Frequency(IDF) are our term weightings approach for this technique. RTF is as follows : $\frac{T_iFreq}{Total}$, $T_i$ denoting a specific term and $Total$ is the total amount of terms in a document. IDF is $log(\frac{D}{d_i})$ $D$ being the count of all documents in the corpus and $d_i$ is the doc count for docs that contain a term of interest. These are multiplied together to get a value that will be used for our Cosine Similarity calculation, this is the final step of document retrieval and how we actually rank the documents. Cosine similarity simply determines the similarity between query and document as follows : $\frac{\sum_i^n A_iB_i}{\sqrt{\sum_i^n A_i^2}\sqrt{\sum_i^n B_i^2}}$ $A$ and $B$ denoting sets of RTF-IDF values that correspond to a document

and query, a bag of words approach is generally used for this sort of thing meaning, we only take the RTF-IDF values (if there is one, otherwise 0) for specific bag of words for the query and document.

Finally we will talk about BERT. BERT in general is just a very advanced language model, but we can train BERT for specific tasks, which is what we did for this. BERT if trained on a question answering dataset will eventually adapt to answer questions given a context passage. We use the documents retrieved in the document retrieval portion as the context passage to answer the given question. Therefore the final process is as such:

1. tokenize corpus
2. build inverted index on tokenized corpus
3. utilize inverted index to retrieve documents that contain keywords in qury
4. rank the retrieved documents with cosine similarity
5. look through each document using the document as the context passage with our fine-tuned BERT until it finds an answer or it runs out of relevant documents

## Implementation

For the Inverted Index an external custom built hash table class was created in java for ease of use. It would work as a HashMap does in java, essentially to create some abstraction so that the insertion and retrieval process was simple to use. Temporary files are created for each document in the corpus, sorted alphabetically, and the temp files are merged down so that the first line of each temp file can be in the memory at once. This is required for creating the posting number, otherwise this would be a memory intensive operation. The query portion is rather straightforward, a bag of words is created from the input words, the RTF-IDF of the query is precalculated, we then find every document that contains one of the words in the query and compile them into a list with no duplicate documents. From here we calculate the cosine similarity between the documents' RTF-IDF values, the queries RTF-IDF values, and

rank them based on this value.

This is where the document retrieval and BERT come together as one. We have a bash script that writes the question and the document names to two separate files, answerdocs.txt and question.txt. From here in the bash script we call a python program that contains the BERT question answering process that will read from these files. The python script is simple in concept, we read in the question and the list of documents that were ranked highest, then we loop through each document to find the context. If the length of the document is longer than 384 words there are some special precautionary steps we must take for BERT to work as our model does not accept an input length of greater than 384 words for context. We split the document up into different 383 length chunks (383 in this case just to avoid any unnecessary errors) and utilize a back-stepping process where we step into the previous chunk of words to gain any context that may have been lost in the split going to the next chunk. This back-step size is 50 words in our case. The splits would look as follows for a length $> 384$ first chunk: words[0:383], second chunk: words[333:716] and so on. 50 is an arbitrary number with no proven reason behind it, we did not have time to test the performance of different amounts and 50 seemed a reasonable amount to not lose most of the context but without risking becoming extremely inefficient. Each chunk is fed into our BERT model alongside the question and this will repeat until an answer is found or we run out of documents to look through.

## Evaluation

Our chosen metric for evaluation was F1 Score. The model was trained on the SQuAD 2.0 and they provided an evaluation script on the website for testing models that were trained on SQuAD. Our model reported an F1 Score of 75.8. This is not nearly as high as the reported scores from our inspiration papers however in regards to the time-span and limitations of our research we believe these results are quite appropriate, we and are happy with them.

## Conclusions

In this paper we specified the techniques for implementing document retrieval with the use of RTF-IDF, Cosine Similarity, and Inverted Indexes and also talked about using BERT for language predictions and fine tuning it for a specific task such as Open Domain Question Answering. Our implementation utilized these specific techniques for document retrieval and utilized a BERT model that was fine tuned on SQuAD. Our reported F1 score is 75.8 which is a respectable number given the time and resource constraints.

Our models accuracy and document retrieval could certainly have been improved if we continued our work. The tokenization that the document retrieval was built on is clunky and not generalized enough and we believe that the results could have been drastically improved in terms of accuracy and runtime. 'ers' 's' 'ies' and etc were not truncated on terms when the inverted index was being built and we believe if these were truncated context of the query could be generalized to more documents and the document retrieval would be more precise, and as a result the question answering would be faster and more precise as well. If we were to continue our work with more time this would be the first thing we would change.

## Work Cited


[1] Bingning Wang Sogou Inc. & Chinese Academy of Sciences, et al. "Document Gated Reader for Open-Domain Question Answering." Document Gated Reader for Open-Domain Question Answering — Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 1 July 2019, dl.acm.org/doi/10.1145/3331184.3331190.

[2] Huawei, Xinyu Zhang, et al. "Answer Complex Questions: Path Ranker Is All You Need." Answer Complex Questions: Path Ranker Is All You Need — Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, 1 July 2021, dl.acm.org/doi/abs/10.1145/3404835.3462942.

[3] Qi, Peng. "Answering Complex Open-Domain Questions at Scale." SAIL Blog, 21 Oct. 2019, ai.stanford.edu/blog/answering-complex-questions/#: :text=This20problem%2C%20called %20open-domain %20question %20answering%20%28open-domain ,QA% 29 %2C%20is%20an%20active%20area%20of%20NLP%20research.

[4] Devlin, Jacob, et al. "BERT: Pre-Training of Deep BIDIRECTIONAL Transformers for Language Understanding." ArXiv.org, 24 May 2019, arxiv.org/abs/1810.04805?source=post_page.