Implementing Open-Domain Question-Answering in a College Setting: An End-to-End Methodology and a Preliminary Exploration

Augusto Gonzalez-Bonorino[1,2], Eitel J. M. Lauría[1,*] and Edward Presutti[3]
[1]*School of Computer Science and Mathematics, Marist College, Poughkeepsie, New York, U.S.A.*
[2]*Data Science & Analytics Dept., Information Technology, Marist College, Poughkeepsie, New York, U.S.A.*
[3]*Enrollment Management, Marist College, Poughkeepsie, New York, U.S.A.*

Keywords: Deep Learning, Natural Language Processing, Transformers, AI in Higher-education, Open Domain Question-Answering, BERT, roBERTa, ELECTRA, Minilm, Information Retrieval.

Abstract: Advances in Artificial Intelligence and Natural Language Processing (NLP) can be leveraged by higher-ed administrators to augment information-driven support services. But due to the incredibly rapid innovation rate in the field, it is challenging to develop and implement state-of-the-art systems in such institutions. This work describes an end-to-end methodology that educational institutions can utilize as a roadmap to implement open domain question-answering (ODQA) to develop their own intelligent assistants on their online platforms. We show that applying a retriever-reader framework composed of an information retrieval component that encodes sparse document vectors, and a reader component based on BERT -Bidirectional Encoder Representations from Transformers- fine-tuned with domain specific data, provides a robust, easy-to-implement architecture for ODQA. Experiments are carried out using variations of BERT fine-tuned with a corpus of questions and answers derived from our institution's website.

1 INTRODUCTION

Open-domain question-answering (ODQA) systems have been a topic of research for many years. Such architectures have been divided into two categories: open-book and closed-book. The former encompasses models that are able to refer back to the corpus of text they were trained on, as well as external knowledge, to output answers. Moreover, the most common framework for such systems is known as a retriever-reader architecture and it was first proposed by researchers at Stanford and Facebook AI in 2017 (Chen et al., 2017). The latter focuses on fine-tuning question-answering (QA) models for precise and rapid answer retrieval from a predetermined dataset containing all possible answers. In this case, the model focuses on finding the answer as efficiently as possible in the curated dataset of question-answer pairs instead of attempting to extract, or generate, it from external knowledge.

We believe that educational institutions could greatly benefit from adopting Natural Language Processing (NLP) technologies such as ODQA systems and intelligent chatbots. Potential benefits include higher efficiency when processing administrative tasks, such as providing information about the institution or guidance on using specific services, tasks traditionally conducted by staff. Furthermore, such a technology can aid in improving the interaction between the students and the different areas of the institution, whether educational or administrative, and in making information easily available to both current and prospective students. Case in point, the ongoing COVID crisis has propelled the use of online environments. The challenge in these environments is the frequent changes in content and content sources. If those virtual environments are enhanced through the use of artificial intelligence, they have the potential of streamlining processes and making them simpler or even seamless. Also, as it relates the prospective student, enrollment management must embrace the current and emerging technologies of social media and online content to maintain its focus on the next generation of students. Online platforms assisted by digital assistants can certainly enhance the recruitment process. Still, and due in part to the rapid evolution of the field of NLP, we notice that such benefits have not been a strong enough incentive to motivate in-

[*]Contact Author.

stitutions to allocate the necessary resources to implement state-of-the-art architectures. For this reason and with the purpose of showing the feasibility of implementing such systems and encourage others to follow a similar path, we have researched ODQA architectures. In this paper we present an end-to-end methodology depicting how to implement such an architecture in an educational institution. The purpose of this work is twofold: a) create a proof of concept of a production-level ODQA system prototype based on a sparse retriever-reader architecture that could be implemented at our institution; b) develop a methodology and guideline, and empirically test different components of a retriever-reader architecture. We have set this framework as our benchmark using the Best Match BM25 (BM25) sparse text retrieval function (Robertson and Zaragoza, 2009) as the retriever, and a reader based on BERT -Bidirectional Encoder Representations from Transformers (Devlin et al., 2019), fine-tuned with domain-specific data. We believe that institutions can make use of this guideline to implement these architectures and take advantage of these promising technologies.

The paper is organized in the following manner: First, we provide a review of the extant existing literature emphasizing the rapid evolution of the field of question-answering since the 1960s and the various techniques and model architectures that have been developed throughout the past decades. We then move on to discuss the core methodology we have designed and implemented at our institution. We follow with the experimental setup, providing an in-depth discussion of the QA dataset, methods, computational platform and software used throughout the experiments, as well as the performance metrics used to assess the performance of the different variations of the retriever-reader architecture tested throughout our research. The experiments include the fine-tuning of the BERT-based reader using a domain-specific QA dataset extracted from the information found in our institution's website. We present and discuss our findings. Finally, we close the paper with comments on the limitations of the research, our conclusions, and future research avenues.

2 LITERATURE REVIEW

Question-answering is a challenging sub-discipline of NLP concerned with understanding how computers can be taught to answer questions asked in natural language. Most QA systems are focused on answering questions that can be answered with short-length texts, also known as factoids (Jurafsky and Martin, 2021). Before the deep learning boom of the 21st century, researchers adopted various approaches to tackle the challenging task of QA. Two paradigms, information retrieval QA (also known as open-domain QA or ODQA) and knowledge-based QA, have been driving the field since the early 1960's. Information retrieval (IR) focuses on gathering massive amounts of unstructured data from different sources (i.e. social media, forums, websites) and encoding those documents in vector representations such that, when a question is received, the retriever component is able to find relevant documents, which are then input to a reading comprehension algorithm (typically referred as the reader) that returns the span of text most likely to contain the answer. Although some QA systems focus solely on the reader component, the reader is given a question and a text fragment with or without the answer, and it returns a resulting prediction answer. It is the combination of both components in the so called retriever-reader architecture that has become the dominant paradigm for ODQA. The reason is simple: whereas in the all-by-itself reader architecture the reader has to deal with the whole document corpus to find an answer to the question, in the case of a retriever-reader architecture the reader uses only the top ranked documents selected by the retriever component. This improves the system efficiency by reducing run-time and increasing accuracy. Knowledge based, or closed-domain QA, on the other hand, consists of collecting structured data and formulating a query over a structured database. One of the first examples of this approach is BASEBALL (Green et al., 1961), a question-answering system that was able to answer questions about baseball games and statistics from a structured database. Until the 80's, knowledge based QA systems grew rapidly by developing larger and more detailed structured databases that targeted narrower domains of knowledge. Nevertheless, since these databases had to be hand-written by experts, they were costly and time consuming. Thus, the focus of research slowly drifted towards ODQA which leverages statistical processing of large unstructured databases.

Traditional approaches for ODQA consisted of the use of sequence-to-sequence models which separate sequences of text (i.e. the question and the corpus of text) via a special token often denoted [SEP]. Research in machine comprehension and question-answering started with one-directional Long-Short-Term-Memory (LSTM) models (Wang and Jiang, 2016). Then, moved onto bi-directional architectures (Seo et al., 2018). Currently, it is embracing the transformer architecture introduced in 2017 by Google researchers (Vaswani et al., 2017) which provides a significant advantage over complex recurrent and convolutional neural networks by focusing solely on attention mechanisms, thereby reducing training time and increasing efficiency. Until then, the inherent sequential nature of those recurrent neural nets limited parallelization within training examples. The transformer architecture replaces the recurrent neural network layer with a self-attention layer used to compute a richer representation of a word in an input sequence given all other words in the sequence at once, leading to simple matrix computations than can be easily parallelized, therefore enhancing performance.

One of the biggest limitations QA-systems faced in the early days was not being able to encode context. This issue was successfully addressed by Google's bidirectional transformer model BERT which was "designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers" (Devlin et al., 2019). Furthermore, the publication of the SQuAD 2.0 dataset (Rajpurkar et al., 2018), developed specifically for fine-tuning question-answering models, enabled researchers to train on both answerable and unanswerable questions. This novel QA dataset, the open-source release of BERT and the development of transfer learning has skyrocketed the interest and resources allocated to the discipline. Note that transfer learning and domain adaptation "refer to the situation where what has been learned in one setting is exploited to improve generalization in another setting" (Goodfellow et al., 2016). Consequently, question-answering -ODQA in particular- has been evolving rapidly. For this reason, we have focused our work on ODQA systems, specifically on retriever + BERT-based reader architectures fine-tuned with domain-specific data.

3 METHODOLOGY

The methodology we describe in this paper is comprised of several steps:
Step 1 - Gather Domain-specific Data: The first step of our project, as in any AI project, is to collect and pre-process data; in this case, the corpus of documents on which questions are formulated and their corresponding answers are produced and delivered by the QA system. For this, we developed a web scraper which pulls the text information contained in every official website from our institution. The scraper extracts all the data and adds it to a json file, from which we then draw out the scraped text for pre-processing. This pre-processing task includes eliminating special characters, stop words, lemmatizing and tokenizing activities. Next, we load all the clean data to a relational database to optimize retrieving performance (on a side note, we initially experimented with MongoDB to exploit its NoSQL document model, but we did not find any significant advantage; in fact, at scale, it under-performed in terms of efficacy and retrieval time).
Step 2 - Choose ODQA Architecture: This step implied some exploration of the extant ODQA existing technologies as well as some trial and error work to find and implement the architecture and its components that best fitted our goals. In the context of a retriever-reader architecture this implies identifying the following items:

a) the information retrieval (IR) system, search engine, or *retriever component*. The system in which the document corpus is stored, encoded and indexed. Encoding in this context means vector encoding; at query time (i.e. when a question is delivered by an end-user) questions are transformed online into vector representations, and the resulting vector-encoded questions are compared by the IR system with the vector-encoded corpus to retrieve a subset of the top ranked documents.
b) the reading comprehension algorithm, or *reader component*, typically a deep learning, transformer-based encoder model such as BERT, that takes the top ranked document(s) fed by the IR system and scans each document to find segments of text - *spans* - that can probably answer the question.
c) the *software platforms* available to implement these components.

Let's consider each of these items in more detail:

Retriever Component: Information retrieval systems uses the vector space model to identify and rank likely documents given a query, in which documents and queries are vector-encoded and matched by similarity using a distance metric in the respective vector space (Salton, 1971). We score (i.e measure similarity) of a document d and a query q by computing the cosine of their respective vector encodings $\mathbf{d}$ and $\mathbf{q}$. If the cosine is 1 there is perfect match.

$$score(q,d) = \frac{\mathbf{q} \cdot \mathbf{d}}{|\mathbf{q}||\mathbf{d}|} \quad (1)$$

IR systems can be classified as dense and sparse depending on the nature of their vector encoding. We considered both types of retriever systems:
Sparse Retrievers: Traditional IR systems encode documents using word frequencies or similar derived metrics such as TF-IDF (Term Frequency – Inverse

Document Frequency) and BM25. These types of IR systems are known as sparse retrievers since words are considered independently of their position and vector encoding is sparse (there is no concept of space embedding based on context). Sparse retrievers are fast both in terms of document encoding (sparse vectors are easy and fast to compute) and query encoding (encoding the query as a vector on the fly) is a trivial task.

The TF-IDF retriever is a popular algorithm used in information retrieval tasks such as finding similar books or documents given a query. It derives its name from the TF-IDF metric, an enhanced replacement of the Bag of Words (BOW) approach to encode text. TF-IDF rewards larger frequency of terms while penalizing terms which occur frequently in documents (e.g. stop words). The metric computes the frequency of a term in a document times the logarithm of the ratio between the number of documents in the corpus and the number of documents where the term occurs. In other words we calculate TF-IDF as follows:

$$\text{TF-IDF}(d,t) = \frac{n_t}{n} \cdot \log \frac{N}{N_t} \tag{2}$$

where n_t is the frequency (count) of term t in document d, n is the total count of all terms in document d, N is the total count of documents, and N_t is the count of documents containing term t.

To compute the distance between document d and query q using TF-IDF we apply Eq.(1) and decompose the inner product as a sum of products over the TF-IDF scores. With some simplification -see (Jurafsky and Martin, 2021) for details, we obtain:

$$score(q,d) = \sum_{t \in q} \frac{\text{TF-IDF}(d,t)}{\sqrt{\sum_{w \in d} \text{TF-IDF}(d,w)^2}} \tag{3}$$

Alternatively, the Best Match 25 ranking metric, also known as Okapi BM25, or just BM25 (Robertson and Zaragoza, 2009), improves retrieval efficiency compared to TF-IDF by considering document length normalization and term frequency saturation (i.e. limiting through saturation the impact that frequent terms in a document can have in the score).

BM25 tunes the term frequency (TF) factor in the TF-IDF formula by adding two parameters: b, which regulates the relevance of normalizing document length; and k, which provides a balance between term frequency and inverse document frequency. The simplified scoring formula for BM25 as implemented in the popular Lucene search engine (Apache Software Foundation, 2021) is defined as follows:

$$score(q,d) \approx \sum_{t \in q} \text{Tuned TF} \cdot \text{IDF}$$

$$\text{Tuned TF} = \frac{n_t \cdot (1+k)}{n_t + k \cdot ((1-b) + b \cdot \frac{n}{avg(n)})} \tag{4}$$

$$\text{IDF} = \log \frac{N}{N_t}$$

A large value of parameter b means that the length of the document matters more (with $b = 0$, the length of the document has no influence on the score). High values of parameter k, in turn, pull the score up when the term occurs more frequently. When $k = 0$ term frequency TF becomes 1, which means that TF-IDF is solely driven by IDF.

In order to efficiently identify the most relevant documents to the input query, the IR system needs a data structure that can relate terms to documents. A typical data structure is an inverted index which lists every unique term that appears in any document, and for each term, all of the documents the term occurs in. This is what indexing means in the context of an IR system. The search is a two-step process: the IR system uses the inverted index to find the subset of documents that contains the terms in the query; only then it proceeds to compute the similarity score on the subset of documents and ranks them to complete the document search.

Dense Retrievers: Sparse retrievers are powerful and fast IR engines, but they have one critical flaw: they lack context. So, as the location of terms in the text is irrelevant, they work in those cases in which there is word-to-word correspondence between the query and the document. Dense retrievers address this issue by replacing sparse vectors with dense vector embeddings. Modern implementations of dense retrievers like Dense Passage Retrieval, also known as DPR (Karpukhin et al., 2020) use transformer models such as BERT -more on BERT in the following paragraphs- to encode the corpus of documents into dense vectors. This can take a long time to process, especially if the corpus of documents is large, as it requires training just like any transformer (deep learning) model. Then, at retrieval time, the query is BERT-encoded on the fly into a dense vector and use to score likely BERT-encoded documents in the corpus using a similar approach as that in Eq(1). Following (Jurafsky and Martin, 2021) notation:

$$\mathbf{q} = \text{BERT}_Q(q)$$
$$\mathbf{d} = \text{BERT}_D(d) \quad (5)$$
$$score(q,d) = \mathbf{q} \cdot \mathbf{d}$$

We considered the use of dense retrievers as part of this work, but the time required to encode the corpus of documents and the lack of evidence of significant improvement when compared to the use of sparse retrievers made us favor the latter approach, at least in this stage of our research. We therefore settled for a sparse BM25 retriever.

Reader Component: Pre-training and transfer learning have been, arguably, the most important breakthroughs of the decade in deep learning. As stated in (Erhan et al., 2010) pre-training ”guides the learning towards basins of attraction of minima that support better generalization from the training data set”. In other words, transformer models are trained on very large amounts of data for a general purpose task, allowing the researcher to then fine-tune such models for specific tasks, like question-answering in this case. The idea of transfer learning branches out from the aforementioned technique. A model pre-trained on natural language, for example, can be reused (i.e. fine-tuned) for a specific task without losing its previous general knowledge, effectively obtaining a powerful model capable of solving complex tasks. BERT-like encoders have therefore become the *de facto* standard for reading-comprehension in ODQA systems. For a list of the BERT-like readers considered in this paper check the section on software platforms.

Software Platforms: It is difficult to create a taxonomy of software platforms that assist in implementing ODQA systems, given the variety of available open-source IR projects, QA frameworks and search engines, and the various ways in which these platforms handle document indexing and storage of indexed documents - in some cases in a vertically-integrated, monolithic fashion, in others as components of a custom-built QA application. As part of this work we investigated a number of the most relevant projects. The list that follows should not be considered as an exhaustive survey of software tools. Instead, it is a reflection of our exploration and the choices we made during this research project to implement a sparse retriever-reader solution for ODQA at our institution:

- *Lucene* (Apache Software Foundation, 2021) is the widely used, high-performance, Apache text search engine project written in Java.
- *ElasticSearch* (Elastic, 2021) is a software package developed by Elastic, based on the Lucene library, that provides a distributed, JSON-based, text-retrieval engine exposing an HTTP API. ElasticSearch is offered under both the Server Side Public License, and the company's own license. Elasticsearch can store both sparse (TF-IDF/ BM-25) and dense representations.
- *FAISS* (Johnson et al., 2017) is a library developed by Facebook research team for similarity search of dense vector representations of text. FAISS is written in C++, can make use of GPUs to speed up the similarity search, and has wrappers for Python.
- *Anserini* (Yang et al., 2017) is a Java-based information retrieval project based on Lucene.
- *Pyserini* (Lin et al., 2021) is a Python information retrieval library that encodes documents in both sparse and dense representations. Sparse retrieval is implemented through integration with the Anserini library. Pyserini uses Facebook's FAISS library to implement dense retrieval.
- *Haystack* (Pietsch, Soni, Chan, Möller, and Kostić, 2021) is a recent open-source framework developed by Deepset.ai for building search systems and ODQA systems that provides a simple interface through which a) documents can be encoded in both sparse (TF-IDF and BM25) and dense (DPR) vector representations and indexed into a variety of data stores; b) a retriever-reader architecture can be easily setup, leveraging a variety of BERT models; c) QA models can be fine-tuned with domain specific data. Deepset.ai has also partnered with HuggingFace (Wolf et al., 2020), a leading NLP startup, to provide state-of-the-art pre-trained models.

As we previously mentioned, the main goal of our research is to provide educational institutions a guideline to implement state-of-the-art ODQA architectures on their campus in a simple and fast way. Our architecture has the structure depicted in Figure 1. We chose the Haystack's API that provides a convenient pipeline to fine-tune and deploy the ODQA architecture. We employed an Elasticsearch server to store our indexed documents to optimize retrieval, a BM25 sparse retriever and a BERT-like reader initially fine-tuned with SQuAD 2.0. We considered three variations of BERT:

- *roBERTa* (Liu et al., 2019), a successor of BERT and trained on a much bigger (10x) dataset, using larger mini-batches and learning rates among other enhancements. All of these differences allow roBERTa to generalize better to downstream tasks and has been the state-of-the-art model for QA tasks in recent years.

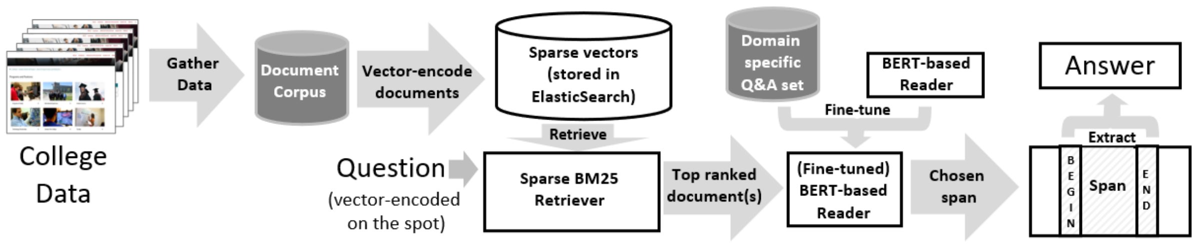

Figure 1: (Sparse) Retriever - Reader Architecture.

- *ELECTRA* (Clark et al., 2020) introduces a new approach to training that produces a model with similar, and sometimes better, results than top-performing transformers using only a fraction of computing resources. ELECTRA replaces the masked language modeling (MLM) pre-training methods used by BERT and other transformers with a more efficient approach called ”replaced token detection”. Strictly speaking, ELECTRA is not really a variation of BERT, but we decided to incorporate it on our experiments due to its novel training approach and potential for QA tasks.
- *Minilm* (Wang et al., 2020) ”distilles” a smaller (student) model from a much larger (teacher) model, like BERT, and with it uses half of the transformer's parameters and computations of the teacher model.

```
reader = FARMReader(
    model_name_or_path= "deepset/
    electra-base-squad2", use_gpu=
    True)

train_dir = "path/to/
    custom_train_dataset"

reader.train(data_dir=train_dir,
    train_filename="
    OurTrainFileName.json",
              use_gpu=True,
              n_epochs=20)

reader.save(directory="
    OurFineTunedModelName")
```

Listing 1: Fine-tuning electra-base-squad2.

Step 3: Fine-tune Reader: Seeking to improve our model's performance, we found the next logical step to fine-tune the reader on our custom QA dataset. For this we created a SQuAD-formatted training dataset using an annotation tool that allowed contributing operators - in this case several student workers- to a) pick a random document from the College website; b) formulate questions on the given document, identifying the answer to the question in the document (first and last word of the answer word of each answer). That is what the BERT-reader does: finds from the corresponding reading passage the beginning and the end of a segment of text, or span, that is most likely to contain the answer to a given question. Listing 1 depicts a sample fragment of Haystack code used to load an ELECTRA model initially trained with SQuAD 2.0 and fine-tune it with our SQuAD2-formatted domain-specific training dataset.

4 EXPERIMENTAL SET UP

In the experiments, we studied the performance of the sparse retriever-reader architecture, considering different versions of the BERT-based reader before and after fine-tuning it using the College website data. The primary goal was to explore the feasibility of the architecture for question-answering at our institution. A second goal was to analyze and measure if, given domain-specific questions, fine-tuning the reader on custom examples would improve its predictive performance.

4.1 Dataset and Methods

We developed a dataset with 594 questions and their respective answers in SQuAD 2.0 format using Haystack's annotation tool. We then proceeded to randomly split the dataset into training and testing partitions using a 80-20 ratio: the training data set was used to fine-tune the BERT-based reader (in those runs where we fine-tuned the model), and the testing dataset was used to measure the performance of the retriever and reader components. Each experiment was performed by varying the random partition seed. We used 30 randomly chosen seeds, creating 30 dataset pairs (training and testing) to perform experiment runs, using 3 different readers (roBERTa, ELECTRA and Minilm). Each of the readers was used in plain-vanilla mode or was fine-tuned using the training datasets. This amounted to a total of 30 runs repeated over 3 x 2 readers, for a total of 180 experiments (for details see section 5 and Table 1).

4.2 Performance Metrics

We collected the metrics listed below. We chose n=5 (the number of top relevant documents identified by the retriever).

Retriever Recall: at n=5 reports how many actual relevant documents (true positives, or TP) were shown out of all actual relevant documents for the question. Mathematically:

$$\text{retriever recall@n} = \frac{\text{TP@n}}{\text{TP@n} + \text{FN@n}} \tag{6}$$

Retriever MAP: for cases where n>1 (the retriever returns a ranked set of relevant documents), average precision, or AveP(q), is a metric that evaluates whether all of the true positive documents retrieved by the system for a given question q are ranked higher or not. The mean average precision, or MAP, is the mean of AveP for all questions q ∈ Q (Q is the total number of questions).

$$\text{retriever MAP} = \frac{1}{Q}\sum_{q=1}^{Q} \text{AveP(q)} \tag{7}$$

Reader Top-n-Accuracy: measures the number of times where the (ground truth) answer is among the top n predicted answers (i.e. the extracted answer spans).

Reader EM: the reader exact match metric, or EM, is an uncompromising, rigid metric; it measures the proportion of questions where the extracted answer span *exactly* matches that of the correct (ground truth) answer at a *per character* level.

Reader F1-score: a measure widely used in binary classification, is in this case computed *over the individual words* in the extracted answer span against those in the correct (ground truth) answer, measuring the average overlap between prediction and ground truth. A good f1 score entails high classifications' accuracy (low false positives and negatives). The scores range from 0 to 1, with 1 being a perfect score.

$$\text{reader F1-score} = 2 \times \frac{\text{recall} \times \text{precision}}{\text{precision} + \text{recall}} \tag{8}$$

Precision and recall in this context are, respectively, the fraction of shared words with respect to the total number of words in the extracted answer span, and the fraction of shared words with respect to the total number of words in the correct (ground truth) answer.

Fine-tuning Execution Time: measured in seconds per epoch. This is of course a relative measure, totally dependent on the power of the available computing platform (see below). Anyway, it is insightful to compare the time needed to fine-tune each of the readers considered in this work.

4.3 Computational Platform

We opted to run most of the experiments over the cloud, using Google Colab(oratory). The Pro+ version yielded a virtual machine (VM) with the following characteristics:

- 8-core Xeon 2.0 GHz
- 54.8 GB RAM
- 167 GB HDD
- 1 Tesla V100-SXM2 16 GB

Additionally, the VM had the capability of running in the background without the need for human intervention which made running multiple experiments non-stop possible.

Running our experiments in Colab limited us to initialize and run ElasticSearch locally each time the notebook ran. A production setting would obviously require a cloud-based or on-premises GPU-enabled platform with access to a retriever (e.g. Elasticsearch) server running in distributed or all-in-one-box fashion.

5 RESULTS AND DISCUSSION

Table 1 displays the assessment of mean predictive performance of each retriever-reader configuration. We use the plain-vanilla reader (no fine-tuning) as a baseline for comparison purposes.

Each experiment was conducted 30 times, each time recording the previously mentioned metrics. The mean predictive performance of each configuration was computed by averaging the 30 outcomes of each metric, together with its standard error. In each variation, fine-tuning the reader on our domain-specific dataset showed significant improvements in the F1 score and accuracy. The following improvements in the reader F1 score were recorded: roBERTa gained 29.57 percentage points on average (49.20% from 19.63%); Electra showed an improvement of 35.49 points (48.78% from 13.29%); and Minilm gained 29.73 (46.25% from 16.52%). In terms of reader accuracy, roBERTa gained 14.02 points (86.69%

Table 1: Results of experiments.

	RoBERTa				**ELECTRA**				**Minilm**			
	fine-tuned		**not ft**		**fine-tuned**		**not ft**		**fine-tuned**		**not ft**	
	Mean	**SE**	**Mean**	**SE**	**Mean**	**SE**	**Mean**	**SE**	**Mean**	**SE**	**Mean**	**SE**
retriever recall	0.90	0.0056	0.89	0.0054	0.90	0.0041	0.90	0.0039	0.89	0.0041	0.89	0.0045
retriever MAP	0.68	0.0076	0.68	0.0050	0.69	0.0058	0.69	0.0049	0.67	0.0055	0.70	0.0050
reader accuracy	86.69	0.5736	72.67	0.6669	85.64	0.4246	68.60	0.5300	84.85	0.5037	70.02	0.4591
reader EM	18.56	0.4307	5.99	0.3700	17.68	0.4759	1.87	0.1786	16.46	0.5250	2.87	0.1950
reader F1	49.20	0.6337	19.63	0.3889	48.78	0.5111	13.29	0.4084	46.25	0.6322	16.52	0.2763
ft exec time	76.59	1.2381			55.42	0.6500			28.79	0.3891		

from 72.67%), Electra 16.84 points (85.64% from 68.60%), and Minilm improved 14.83 points on average (84.85% from 70.02%). Overall fine-tuned roBERTa slightly outperformed the fine-tuned versions of both ELECTRA and Minilm.

The retriever component performed very well, with recall and MAP values of 0.90 and 0.68 respectively. We also computed the time in seconds per epoch it took each model to fine-tune on our 594 questions dataset, recording the following average execution times: a) roBERTa - 76.59 s/epoch; b) ELECTRA - 55.42 s/epoch; c) Minilm - 28.79 s/epoch. As can be seen, ELECTRA and Minilm took considerable less time per epoch than roBERTa, a much larger model, with Minilm's execution time being close to one third of roBERTa's execution time. This could have an impact if the training dataset is large and training time is a relevant factor in the models' deployment.

Interestingly, we found that, despite roBERTa's robustness and significantly greater size. it did not perform that much better than ELECTRA, which showed the greatest gains in performance from fine-tuning (35.5 points in F1 and 16.8 points in accuracy). This suggests that ELECTRA's novel training approach may provide a significant advantage over bigger models such as roBERTa, specially when being considered for production, given its similar accuracy and faster training time.

6 CONCLUSIONS AND FUTURE WORK

There is a plethora of benefits to be garnered by educational institutions from applying these technologies, such as helping students (both incoming and current) and families learn more about the institution, guide students through the process of common tasks such as enrolling in classes or deciding on a major, to name a few. More importantly, it would help decrease the number of administrative tasks currently conducted by staff, opening the door for allocating those resources more efficiently. As a result of the automation of the tasks involved in assisting students and their families there is the added benefit of being able to track what questions are being asked, how well the AI replied to the question and the context of the question. Leveraging these statistics can help an institution better focus staff on tasks and issues that are most important and determine the areas of improvement for the supporting systems and the ODQA application. Further, it is conceivable that one could create an evolutionary process that would automatically improve the performance of the ODQA and by extension the student support services.

Still, we do understand that under-budget and time-constrained administrators may find such implementation challenging. In this paper, with the intent of minimizing the resources required, we have described a methodology for the implementation of ODQA system by abstracting the process into general components, and provided a detailed description of the extant literature. The goal of this research is not centered on tuning the models for optimal performance: a larger dataset with several thousand question-answer pairs would improve the performance metrics, and would open a new research avenue: tracing performance metrics vs number of training instances in the fine-tuning process. This is certainly one aspect of this project which could be improved.

We centered our efforts on sparse retrievers as we did not find considerable benefit during the initial tests that would justify the use of dense representations within the time constraints of this work. Still, this is currently a hot topic of research, and therefore such configurations should probably be revisited when considering the implementation of the retriever component.

Moreover, other novel approaches could also be considered in future work. For example, recently Facebook researchers introduced a generative approach to Question-Answering, called RAG (Retrieval-Augmented Generation) architecture (Lewis et al., 2021). This alternative architecture implements a dense retriever with a variation of BERT as a reader. Their innovation lies in the answering mechanism: in addition to retrieving the best documents, RAG generates answers -instead of retrieving the best span of text- and complements the process by evaluating relevant words within the retrieved documents related to the given question. Then, all individual answer predictions are aggregated into a final prediction, a method which they found to enable the backpropagation of the error signals in the output to the retrieval mechanism, and with it potentially improving the performance on end-to-end systems such as the one described in this research. It would be therefore interesting to gauge the performance of a RAG architecture, compared to the architectures depicted in this paper.

Finally, we acknowledge that the methodology depicted in this research work could be applied in different industry sectors, but it does provide a roadmap for researchers and practitioners in artificial intelligence for higher education to implement open domain question-answering systems. In this regard, this conference is the right venue to promote a discussion on these topics.

ACKNOWLEDGEMENTS

The authors would like to thank Marist College IT department for their sponsorship and support. We would also like to thank the Data Science and Analytics team for their collaboration throughout this research project. A special mention goes to Maria Kapogiannis, Ethan Aug, Michael Volchko and Jack Spagna who worked on the creation of the set of questions and answers used to fine-tune the models.

REFERENCES

Apache Software Foundation (2021). Lucene. https://lucene.apache.org.

Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading wikipedia to answer open-domain questions.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. _eprint: 1810.04805.

Elastic (2021). Elasticsearch. https://github.com/elastic/elasticsearch.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(19):625–660.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*, page 526. MIT Press. http://www.deeplearningbook.org.

Green, B. F., Wolf, A. K., Chomsky, C. L., and Laughery, K. (1961). Baseball: an automatic question-answerer. In *IRE-AIEE-ACM '61 (Western)*.

Johnson, J., Douze, M., and Jégou, H. (2017). Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

Jurafsky, D. and Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3rd, draft edition. https://web.stanford.edu/~jurafsky/slp3/.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.

Lin, J., Ma, X., Lin, S.-C., Yang, J.-H., Pradeep, R., and Nogueira, R. (2021). Pyserini: An easy-to-use python toolkit to support replicable ir research with sparse and dense representations.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Pietsch, Soni, Chan, Möller, and Kostić (2021). Haystack (version 0.5.0). https://github.com/deepset-ai/haystack/.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad.

Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389.

Salton, G. (1971). *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., USA.

Seo, M., Kembhavi, A., Farhadi, A., and Hajishirzi, H. (2018). Bidirectional attention flow for machine comprehension.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. _eprint: 1706.03762.

Wang, S. and Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. (2020). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Huggingface's transformers: State-of-the-art natural language processing.

Yang, P., Fang, H., and Lin, J. (2017). Anserini: Enabling the use of lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1253–1256, New York, NY, USA. Association for Computing Machinery.