Analysis and extension of BERTserini open-retrieval question answering framework

Armenia Sara
Politecnico di Torino
s279590@studenti.polito.it

Lacriola Francesco
Politecnico di Torino
s292129@studenti.polito.it

Tomasello Noemi
Politecnico di Torino
s272600@studenti.polito.it

***Abstract*—Given a question posed by a user in natural language, Question Answering is a NLP task which aims at giving a relevant answer, or abstaining if the question cannot be answered based on the provided context. Interest in QA field increased considerably in recent years since its applications are many and varied: from chat-boxs to search engines.**

We address open-domain QA, presenting in particular a replication study of BERTserini, Yang et al [2], an end-to-end question answering system composed by two main components: an information retrieval and a reader. We replace BERT, the actual reader in BERTserini architecture, with DistilBERT and RoBERTa, studying their impact on the model in terms of overall performances. As expected, the fine-tuning of DistilBERT turns out to be faster, at the cost of the performance. Best results in terms of exact match and F1 are obtained using RoBERTa thanks to its improved pre-training strategy.

We also introduce EasyNMT, the state-of-the-art Neural Machine Translation, on top of the retriever, making a user able to ask a question and obtaining an answer in more than 150 different languages.

Aware that existing neural models tend to fail when they try to generalize and acquire transferable reading skill, we trained BERTserini on a further dataset, TriviaQa, exploiting all the positive aspects of data enrichment. A surprising improvement was reached with just a small portion of the data-set.

Our code is available on GitHub at `https://github.com/frank-lacriola/BERTserini`,

***Index Terms*—Anserini, BERT, DistilBERT, RoBERTa**

I. Problem Statement

Question answering (QA) is one of the most impacting task in natural language processing. Its spread in the last years is testified by the huge variety of applications it is used in: QA systems can be currently found in search engines and phones conversational interfaces, such as chat-bots or AI assistants.

Question answering is a specific type of information retrieval. Given a set of documents, it attempts to find out the correct answer to the question posed in natural language by humans.

QA systems are either closed domain or open domain. A domain can be seen as a manifold in a high-dimensional variety space consisting of many dimensions, which are the features of our data. Therefore, in closed-domain QA, questions and answers relate to specific domains and exploit domain-specific knowledge, whilst in open-domain QA they relate to broad knowledge. Lewis et al. [9] identify three classes of questions that an Open Domain QA system should be able to answer in increasing order of difficulty:

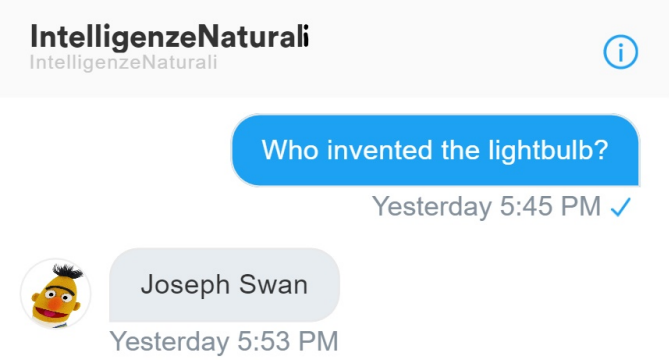


Fig. 1. Question-Answering result obtained from our pipeline.

1) The most basic behaviour is to be able to reliably recall the answer to a question that the model has seen at training time.
2) A model should be able to answer novel questions at test time and choose an answer from the set of answers it has seen during training.
3) A strong system is supposed to answer novel questions which have answers not contained in the training data.

Two major techniques are exploited to develop these systems: Reading Comprehension and Open-Retrieval QA.

Reading comprehension is the ability to read a piece of text and then answer questions about it. This can be seen as a sub-problem of open-retrieval QA as it assumes that we have access to the "gold" paragraph that contains the answer. The task is to discover the shortest fragment of text in the document that answers the user's query, which usually is the last phase of "answer extraction". In both settings, answers are commonly represented as a minimal span. Standard approaches for reading comprehension rely on pre-trained models such as BERT [3], which can be eventually fine-tuned on task-specific datasets like SQuAD [6]. The model is provided with the question and candidate paragraph as input and is trained to predict whether the question is answerable and whether each token is the start or end of an answer span.

Open-retrieval QA, instead, focuses on the most general setting in which we first need to retrieve relevant documents from a large corpus. Given a question x and a ground truth answer span y, the context passage containing the true answer is labelled as $z \in Z$, where Z is the external knowledge

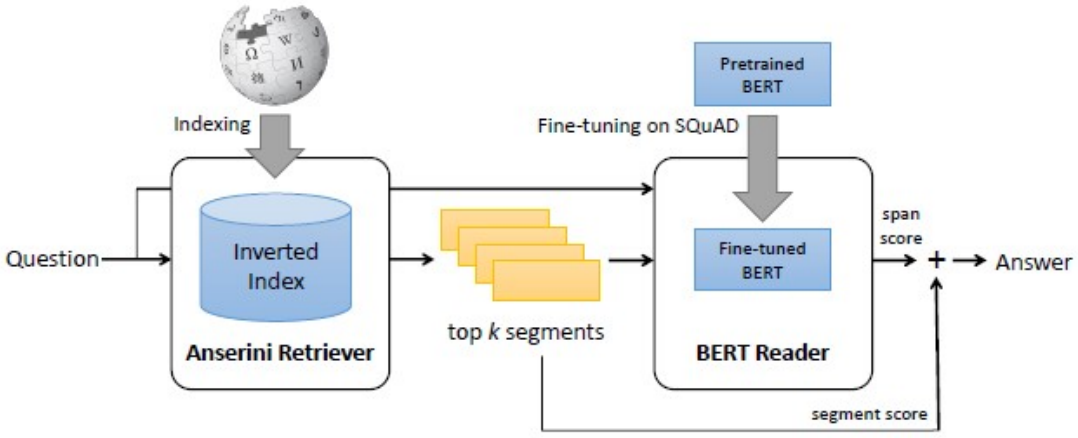


Fig. 2. BERTserini architecture.

corpus. Wikipedia is a common choice for such an external knowledge source. Information retrieval methods are used to retrieve the relevant context passages, which are processed to identify the most relevant answer with machine reading comprehension systems. For these reasons, these kinds of frameworks are made up of a retriever and a reader as building blocks. This was first proposed in [10]. The most widely used IR approaches for implementing the retriever is to use systems that rely on non-learning-based TF-IDF features or dense embedding vectors of text produced by neural networks. To solve the reading comprehension task, instead, the reader model extract an answer for a given question from a given context document. Before the advent of BERT, this was mainly implemented with Bi-directional LSTM.

Yang et al. integrate BERT with Anserini, an open-source IR toolkit built on top of the popular Lucene search engine, to create the BERTserini QA system [2]. The proposed framework consists in an End-to-End Open-Domain Question Answering system, where end-to-end stands for the need of information retrieval over a large document collection and machine reading comprehension of the retrieved passages. We also adapted the framework to use alternative models as readers, in particular the lightweight DistillBERT, a small and fast Transformer model trained by distilling BERT base, and RoBERTa, which improves BERT with a robustly optimized method for pretraining. Moreover, we use EasyNMT to add the possibility to translate the user's question when this is not posed in English, making the framework multi-lingual.

Our study consists also of a data enrichment: we inspect the contribution of a first fine-tuning stage on TriviaQA, like proposed in [3], before fine-tuning the BERT model on SQuAD.

II. Methodology

To address the Question Answering problem we implement BERTserini, which is an end-to-end QA system composed by two main components: the open-source Anserini information retrieval and BERT. The architecture of BERTserini is shown in Figure 2.

A. ANSERINI retriever

When a retriever model is supplied by a question, the question is encoded in a way that it could be compared to the context vectors into the same vector space in order to retrieve the most relevant contexts. These contexts are successively passed to the reader. The retriever is the most critical component in the process, since the success of the overall system is highly dependent on first providing a correct passage to read through. As Yang et al [2] show in their paper, we use a single-stage retriever able to directly identify segments of text from the data-set provided and feed the BERT reader. Information Retrieval systems do not operate on original text documents, but on pre-processed documents using indexing techniques. Inverted index technique is used in this context and returns an index data structure that allows an efficient, full text searches. It builds mapping from content, such as words, to its location in a document or set of documents. In essence, the dictionary of terms is built by defining the set of unique words in the corpus. Each term is associated to a list that reports which documents the term occurs in. When Anserini Retriever receives as input a question, the searcher is in charge of reading and searching indexes, to find candidate contexts for the answer to the posed question. In our implementation, we make use of Lucene – IndexSearcher.

B. BERT reader

A reader model, given a question and context, attempts to identify the best text span from the context which answers the question. The language representation model selected for our purpose is BERT, since the pre-trained BERT model can be adapted without drastically changing its architecture and become the state-of-the-art for different tasks. Following this direction, the architecture is kept unchanged, with the only difference the final softmax layer over different answers spans is removed to allow comparison and matching of results from different segments. More specifically, the pre-trained model involved is bert-base-uncased. Uncased means it does not make a difference between capital or lowercase letters. The

model is pre-trained on a large corpus of English data (which is BookCorpus, a dataset consisting of 11,038 unpublished books and English Wikipedia, excluding lists, tables and headers) in a self-supervised fashion: it is pre-trained on the raw texts only, with no humans labelling them in any way, with an automatic process to generate inputs and labels from those texts. Pre-training is done taking into account two main objectives:

- *Masked language modeling*: taking a sentence, the model randomly masks 15% of the words in the input (of the selected tokens, 80% are replaced with [MASK], 10% are left unchanged and 10% are replaced by a randomly selected vocabulary token), then run the entire masked sentence through the model and has to predict the masked words. The MLM objective is a cross-entropy loss on predicting the masked tokens. In this way the model learns a bidirectional representation of the sentence.
- *Next sentence prediction*: it is a binary classification loss for predicting whether two segments follow each other in the original text. In essence, the model concatenates two masked sentences (positive examples are created by taking consecutive sentences from the text corpus. Negative examples are created by pairing segments from different documents. Positive and negative examples are sampled with equal probability) as inputs during pre-training. The model has to predict if the two sentences were following each other or not.

To make the model suitable for question answering task, we fine-tune it on the training set of SQuAD. Reader's parameters values are the default ones. The reader selects the best text span and provides a score. The preferred answer is the result of the linear interpolation of the reader and retriever score:

$$S = (1 - \mu)S_{Anserini} + \mu S_{Bert} \tag{1}$$

where $\mu \in [0, 1]$ is a hyper-parameter.

C. DistilBERT

DistilBERT is a pre-trained general-purpose language representation model, which can be fine-tuned with good performances on other tasks, question answering included, like the father version BERT. As the name suggests, DistilBERT is the distilled version of BERT, a model built with the purpose of being smaller, faster, cheaper and lighter. Its size is actually reduced by 40%, it is 60% faster, keeping the 97% of its language understanding capabilities. DistilBERT refers to the concept of "knowledge distillation": it is a compression technique in which a compact model, the student, is trained to reproduce the behaviour of a larger model, the teacher, in this case BERT. The student model is pre-trained on the same corpus in a self-supervised fashion, using the BERT base model as a teacher. More precisely, it was pre-trained with three objectives:

- *Distillation loss*: the model is trained to return the same probabilities as the BERT base model;
- *Masked language modeling*: this is part of the original training loss of the BERT model;
- *Cosine embedding loss*: the model is also trained to generate hidden states as close as possible as the BERT base model.

In this way, the model learns the same inner representation of the English language than its teacher model, being at the same moment faster for inference or downstream tasks.

In [3], DistilBERT has the same general architecture as BERT, the only difference is that the token-type embeddings, i.e. the vocabulary IDs for each of the tokens, and the pooler, that is the layer that applies a linear transformation over the representation of the first token, are removed. Furthermore, the number of layers, 12 in BERT, are halved.

D. RoBERTa

RoBERTa model is born as a replication study of BERT pretraining, with a specific focus on the effects of hyper-parameter tuning and training set size, which have a significant impact on the final results. The modifications characterizing RoBERTa with respect to BERT are the following:

- the model is trained longer, with larger batches, considering a bigger dataset. The dataset in question is CC-NEWS, which contains English language news articles from news sites all over the world. The choice of using a novel, bigger dataset confirms that using more data in pre-training phase further improves performance on downstream tasks. Training with large batches ($BERT_{BASE}$ 1M steps with a batch size of 256 sequences, RoBERTa 125K steps with a batch size of 2K sequences or 31K steps with a batch size of 8K) improves perplexity for the masked language modeling objective, as well as end-task accuracy;
- the training is made on longer sequences without next sentence prediction objective. BERT is pre-trained using two concatenated document segments, but recent work has questioned the necessity of the NSP loss. Different alternative training formats demonstrate to improve downstream task performance, one above all the full-sentences one: each input is packed with full sentences sampled contiguously from one or more documents, such that the total length is at most 512 tokens. Input may cross document boundaries.
- change of the masking pattern applied to training data. The original BERT implementation performed masking once during pre-processing. This masking type is called "static mask". In RoBERTa dynamic masking is preferred: the masking pattern is generated every time the model is fed with a sequence, resulting a critical aspect in pre-training for more steps or with larger datasets.
- use of larger byte-level Byte-Pair Encoding (BPE). It is a hybrid between character- and word-level representations that allows handling the large vocabularies common in natural language corpora. Instead of full words, BPE relies on subwords units, which are extracted by performing statistical analysis of the training corpus. A clever implementation of BPE, proposed in [5] uses bytes instead of unicode characters as the base subword units.

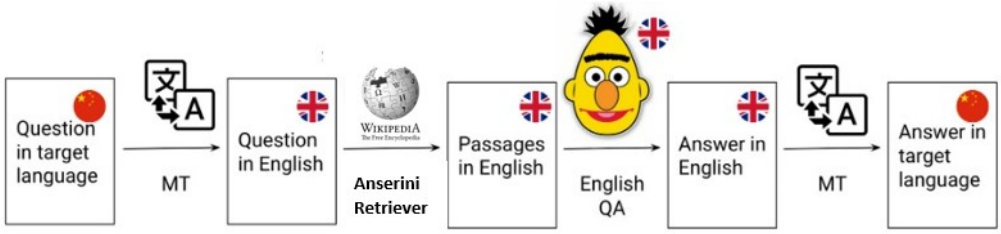


Fig. 3. With EasyNMT the model is able to translate a question in English and to return a relevant answer in the desired language.

The original BERT implementation uses a character-level BPE vocabulart of size 30K, RoBERTa uses BPE vocabulary containing 50K subwords units. This adds approximately 15M parameters for $BERT_{BASE}$.

Performance can be substantially improved by training the model using all aforementioned arrangements. When the best RoBERTa model is evaluated on the SQuAD benchmark (the model is fine-tuned with the provided SQuAD v2.0 training data, additionally classifying whether a given question is answerable or not), it sets a new state-of-the-art, reaching F1 = 89.8.

E. Machine Translation

Machine Translation (MT) is the task of automatically converting one natural language into another, preserving the meaning of the input text, and producing fluent text in the output language. Three different approaches can be followed to implement MT: rule or dictionary-based, statistical and neural. We adopt neural machine translation which is able to reach better contextualization and results and it costs a limited human engineering effort. Hence, on top of the retriever we use EasyNMT to address non-English questions. The model can deal with more than 150 different languages and it is the state-of-the-art Neural Machine Translation. A schema of the process is represented in Figure 3.

F. Data Enrichment

To make our model more robust, flexible and to obtain more precise and accurate answers we adopt data enrichment strategy.

Data enrichment is simply the process of enhancing existing information by supplementing incomplete data. In essence, we fine tuned the model using two different datasets: TriviaQa first and then SQuAD. Pre-trained models fine-tuned on a single domain often generalize poorly. Instead, fine-tuning a model jointly across multiple QA datasets leads to better generalization to unseen domains.

The main challenge we address is that different datasets often have different formats, which makes it difficult to train a joint model without task-specific engineering. Explanation of how we handle this problem is detailed in Experiments.

III. Experiments

We decide to try out two kinds of experiments. The first one aims to validate the pipeline with different models like DistilBERT and RoBERTa. The other one implement a data enrichment task, fine-tuning the model on TriviaQA and then on SQuAD. Finally, a simple multi-lingual pre-trained model is added to allow the system to translate the question in other languages to English; vice versa for the answer.

A. Data

As already mentioned, the pipeline consists of two main parts: retriever and reader. To implement the system as well as the experiments the data used by both components of the pipeline remain unchanged.

The retriever uses Lucene indexes to extract the contexts for our question-answering pipeline. As done in the original paper [2], we use documents based on the English Wikipedia. The Pyserini [7] module, which is the Python version of the original Answerini module written in Java, provides different kinds of searchers: Sparse Retrieval, Dense Retrieval, and Hybrid Sparse-Dense Retrieval. It also provides several pre-built indexes to use with each of these retrievals. However, due to the limited resources provided on Google Colab, we strictly use the Dense Retrieval with the indexes named "*enwiki-paragraphs*" of about 16GB or "*wikipedia-dpr*" of about 8GB. Both indexes are composed of English paragraphs extracted from Wikipedia. Note that in the latter, to reduce indexes sizes, positions are not indexed (so, no phrase queries), and document vectors are not stored (so, no query expansion). The Sparse Retrieval uses a Lucene Searcher, built on the Apache Lucene search engine. Lucene's index is an inverted index. Hence, it lists, for a term, the documents that contain it, instead of having documents listing terms.

The Reader, instead, is fined-tuned on SQuAD (Stanford Question Answering Dataset).

During the experiment we also perform data enrichment with the TriviaQA dataset.

1) SQuAD: It is the leading dataset used to perform Question-Answering. It is made of 100,000+ question-answer pairs posed by crowd-workers on a set of Wikipedia articles. [6]. In SQuAD an input consists of a question and a paragraph for context. The goal is to find the span of text in the paragraph that answers the question.

2) TriviaQA: TriviaQA [8] is a reading comprehension dataset obtained from 14 trivia and quiz-league websites. It contains over 650K question-answer-evidence triples, that are

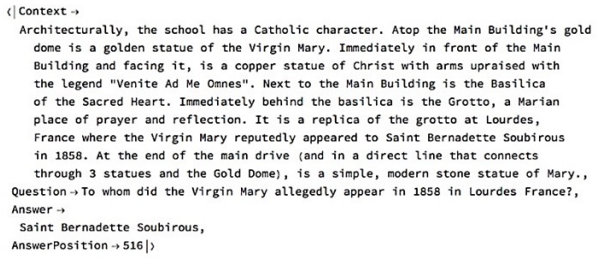
	EM	F1	Training Time
BERT	78.74	86.68	4h.50m
DistilBERT	72.59	81.52	3h.20m
RoBERTa	85.40	91.63	5h.05m

TABLE I
RESULTS: FINETUNING ON SQUAD

	EM	F1
BERTserini (BERT on SQuaD)	25.0	29.76
BERTserini (DistilBERT on SQuAD)	24.3	28.19
BERTserini (RoBERTa on SQuAD)	29.81	32.33
BERTserini (BERT on TriviaQA + SQuAD)	25.0	33.09

TABLE II
BERTSERINI RESULTS

derived by combining 95K Trivia authored question-answer pairs with on average six supporting evidence documents per question. It is the first dataset where full-sentence questions are authored organically, independently of a NLP task. The supporting evidence documents per each question collected from both Wikipedia and more general Web pages. This helps overcome potential bias in the data and provides a wider variety of topics. However, there is no guarantee that the answer can be found in the documents. Hence, TriviaQA refers to the documents as distant supervision. Specifically, the general Web resources are expected to be more redundant, therefore the question is considered as an answer-document tuple. While, the Wikipedia documents do not repeat facts, so each question is considered as a single data point.



Question: The Dodecanese Campaign of WWII that was an attempt by the Allied forces to capture islands in the Aegean Sea was the inspiration for which acclaimed 1961 commando film?
Answer: The Guns of Navarone
Excerpt: The Dodecanese Campaign of World War II was an attempt by Allied forces to capture the Italian-held Dodecanese islands in the Aegean Sea following the surrender of Italy in September 1943, and use them as bases against the German-controlled Balkans. The failed campaign, and in particular the Battle of Leros, inspired the 1957 novel **The Guns of Navarone** and the successful 1961 movie of the same name.

Fig. 4. Question-answer examples from SQuAD v1.1 and TriviaQA.

B. Metric

We exploit the HuggingFace method to load the official scoring for the SQuAD dataset, which is used also as the official evaluation metrics for the TriviaQA dataset. This consists of two metrics, typically used to evaluate question-answering tasks:

- Exact match: is the percentage of the predictions that strictly match the ground truth answer. For correct answers the EM will be 1, $EM = 0$ otherwise.
- (Macro-averaged) F1 score: computed over the individual words in the prediction against those in the True Answer. This metric equally weight precision, i.e. the ration between the length of shared words and the total number of words in the predicted answer, and recall, i.e. the ratio between the number of shared words and the total number of words in the ground truth.

C. Fine-tuning Details

The fine-tuning has been performed by exploiting the Trainer API provided by the HuggingFace library. As training arguments we have:

- learning rate: $3e-5$
- train-eval batch size: 16
- seed: 42
- max_length = 384
- stride = 128

Due to the limited resources, we decided to use only the 80% of the SQuAD dataset and fine-tuning for 1 epoch. Typically on a Tesla K80 GPU the training lasts around 5 hours. We exploited checkpoints to resume the training process whenever it was necessary. Moreover, we enabled the parameter f16=True, to speed up the training time. This enables the use of AMP (automatic Mixed Precision) that allows to train the model using the mixed precision format. AMP helps with math-intensive operation, memory-limited operation, and it reduces memory requirements.

D. Model variation

As part of our experimental phase, we test DistilBERT and RoBERTa models to evaluate the differences in performance with respect to BERT. All the hyper-parameters, and the percentage of the SQuAD dataset used are the same listed in the fine-tuning section, except for DistilBERT's *train-eval batch size*, which is set to 32. In this way, we can compare the results for each model.
The results of the finetuning on for each model can be seen in Table I. As expected, we get better results with RoBERTa and slightly worse results for DistilBERT with respect to the BERT results for EM and F1 scores. On the other hand, DistilBERT has a significantly lower training time with respect to the other two models as expected.

E. Data Enrichment

TriviaQA dataset has been used to perform the data enrichment experiment. Since the dataset has a large amount of data and we deal with limited resources, we are required to use only 5% of the dataset from the Wikipedia subsection to fine-tune the BERT base model for one epoch. The hyper-parameters are the same for fine-tuning the BERT base model

on SQuAD. With this configuration and a Tesla GPU K80, the fine-tuning takes $\sim 5h$ to complete.
TriviaQA has a different format than SQuAD, so we need to reformat the dataset to be able to use the Trainer API as we do for SQuAD. In particular, TriviaQA consists of six columns: 'question', 'question_id', 'question_source', 'entity_pages', 'search_results', 'answer'. For question answering scope, just context, answer, start and end position are required. The context is already contained in the 'question_source' field, for the answer we look at the answer value, instead, start and end position, are taken from the position of the answer in the context.

F. Machine Translation

Finally, we decide to insert a machine translation model in the two-stages pipeline to address the non-English question cases. When the question is given to the retriever, we first check if the language is English. In case is detected a different language, the question is translated into English and fed as normal to the retriever. In the end, we translate the answer found from English to the original language detected for the question, exploiting the same language model. Naturally, the drawback of this method is that the retrieval of contexts will always be in English since we do not have at disposal several multi-lingual indexes for the *LuceneSearcher*.

G. Results

Evaluate our pipeline using Google Colab has been inevitably challenging due to the time limit on the usage and ram available. Therefore, we decide to test the pipeline with a small set of examples from the SQuAD validation set and compare our results on them. We make sure to set a seed to randomly select the same examples for all the models to have comparable results. During the retrieval, we select $k = 10$ contexts to feed the reader. While the weight μ in the selection of the best answer is set to 0.5, to balance out the retriever and reader scores. As we can see in Table II, the results using the pipeline mirror the ones obtained during fine-tuning. RoBERTa improved the F1 and EM scores in comparison with BERT, while DistillBERT still achieving good performance, does not show any speed-up at inference time. Moreover, we can see that the best results obtained are from BERT finetuned on both TriviaQA and SQuAD.

IV. Conclusion and takeaways

The aim of the project is to create a valid replication study of BERTserini. The retriever has a crucial importance during evaluation, since non-relevant contexts can undermine the success of the overall system, making the performances drop drastically. Changing the reader we can appreciate its effect on the entire pipeline, in term of training speed and accuracy reached. It is easy to notice, that a more powerful hardware is needed to be able to perform finetuning with more epochs and using the whole datasets. Whit this in mind, results confirm our first hypothesis: DistilBERT requires an affordable training time, still achieving satisfying performances. Not without reasons, it has proven to be tailored for environments that are boxed in terms of resources, like edge devices. On the other hand, RoBERTa, whose training time is not negligible, showed its power during fine-tuning, outperforming other models. The most interesting and unexpected achievement is the significant increasing in the accuracy after computing data enrichment on a small portion of TriviaQA, specifically the Wikipedia subset. This result is promising: having the possibility to train on the entire datasets the pipeline could be a valid starting point for usage in production environment.

References

[1] Pyserini- https://github.com/castorini/pyserini
[2] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. End-to-End Open-Domain Question Answering with BERTserini
[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
[4] Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter
[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi,Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach
[6] Alec Radford, Karthik Narasimhan, Time Salimans,Ilya Sutskever. Improving language understanding with unsupervised learning.
[7] Pranav Rajpurkar and Jian Zhang and Konstantin Lopyrev and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text
[8] Mandar Joshi, Eunsol Choi, Daniel S. Weld, Luke Zettlemoyer. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension
[9] Patrick Lewis, Pontus Stenetorp, Sebastian Riedel. Question and Answer Test-Train Overlap in Open-Domain Question Answering Datasets.
[10] Danqui Chen, Adam Fisch, Jason Weston, Antoine Bordes. Reading Wikipedia to Answer Open-Domain Questions