An adaptable method for developing an Open-domain Question Answering system

Sara Piscitelli1,* , Sara Cardarelli1 and Alessandro Nicolosi1

1Lab of Applied Artificial Intelligence, Leonardo Labs, Leonardo S.p.A., Via Tiburtina km 12400, 00131 Rome, Italy

Abstract
Open-domain QA has arisen as a natural language processing research subject to answer users’ inquiries in natural language from vast unstructured text collections. Among the most important business areas, using a corpus of unstructured documents as a source of information to answer a question is just one of the many benefits of working in an open-domain setting, which also allows for access to external knowledge even if the information related to the question is not uniquely identified in the sources. Thus, any procedures involving extracting information and data from available online sources and text documents may be enhanced with the help of open-domain question-answering systems. In this paper, we propose a method for developing an open-domain question-answering system by retrieving documents from a knowledge corpus comprising external and internal documents in response to input query and then extracting the answer using an instruction fine-tuned language model, following a “zero-shot” approach. While conceptually simple, this approach can be used as a flexible framework to develop an open-domain question-answering system efficiently.

Keywords
nlp, open domain question answering, language models, information retrieval

1. Introduction
Open-domain question answering (OpenQA) has emerged as a natural language processing (NLP) research field aimed at answering questions in natural language from retrieved extensive unstructured text collections in response to users’ questions. In leading business domains, an open-domain setting allows using a corpus of unstructured texts to answer a query and access external knowledge, even if the information related to the query is not univocally identified in the sources. Hence, open-domain question-answering systems may enhance data extraction from open web sources and text documents (e.g., internal documents, emails, online reviews, and other sources). Most of the existing OpenQA systems [1], [2], [3], [4], [5], [6] follow two-stage Retriever-Reader approaches. In the first stage, relevant information in response to an input query is retrieved. Then in the second stage, a language model answers the question from the retrieved corpus of documents. In this paper, we propose a method to develop an OpenQA by retrieving relevant documents based on a knowledge corpus drawn from both open sources and internal documents to answer input questions, and then, extracting the answer using a modern instruction fine-tuned language model. Developing an OpenQA system involves several challenges. When you have access to a vast external knowledge source, the first challenge is finding a way to retrieve documents that contain useful information to answer the question. It is necessary to develop a strategy for filtering information. Several papers introduce a reranking component as part of a multi-step retrieval strategy to enhance more confidentiality about the retrieved results [7], [8], [9]. Given an initially retrieved passage list, the reranker component scores them, enabling the selection of the most suitable answer aggregating evidence from multiple passages. When developing our system, we employed a fine-tuned instruction language model as an alternative, delegating both the question answering and aggregation-answering tasks to it. It has been proven that modern language models, particularly large ones with billions of parameters, are competitive in many NLP tasks [10], mainly due to their in-context learning behavior [11] that allows them to perform several tasks by conditioning on input-output examples, without optimizing any parameters or performing any gradient updates. Since the employment of models with billions of parameters are both time-consuming and costly to train and query, we opted for a pre-trained instruction-finetuned language models (flan-t5-large [12]) used with a “zero-shot” strategy, meaning no demonstrations are provided at inference time.

Our approach proposes an OpenQA pipeline consisting of three steps. In the first step, the retriever component uses transformer models to retrieve documents relevant to a specific input question. In the second step, the reader component is realized with an instruction fine-tuned language model. Due to its instruction-finetuning process, several prompts can be used to perform multiple natural
language processing tasks including question-answering. In the last step, the language model is again used for aggregate multiple answers into one, but with different instructions to allow it to combine the previous answers. Overall, our contributions are as follows:

• 1) We propose an OpenQA framework that utilizes a “zero-shot” approach by using transformer models in both the retriever and reader.

• 2) The approach is not domain-specific, but can be easily adapted to different use cases and models can be trained on a specific dataset to achieve better performance.

• 3) Using an instruction fine-tuned language model as a reader component is effective for extracting a unique answer from multiple documents.

2. Related Works

Modern OpenQA systems use a "Retriever-Reader" approach: a retrieval component finds relevant passages from a large corpus and a reader extracts the answer using machine reading comprehension. As far as we know, one of the earliest systems using this approach was suggested by Chen et al. in 2017 [13] with a document retrieval Question Answering (DrQA) system using a tf-idf retriever and recurrent neural network reader to match similar documents related to a query and deliver the final answer. The positive outcomes of this approach have resulted in an emphasis on the connection between OpenQA performance and the retriever’s capacity to locate pertinent text excerpts for a given query. As a result, Wang et al. [7] added a “ranker” to improve retrieved outcomes by ranking passages. In 2019, several works proposed improvements to this architecture. Wei Yang et al. [6] suggested a system consisting of an Anserini retriever, a bag-of-words retriever that directly identifies text segments from Wikipedia, and a BERT reader to perform extractive QA by selecting the text span where the answer can be found. Seo et al. proposed a Dense-Sparse Phrase Index [2] as a retriever based on a transformer BERT embedding of each sentence to encode syntactic or semantic information of the phrase with respect to its context. This leads to an improvement in efficiency and performance compared to the previous system DrQA[13], suggesting that using the most recent transformer model [14] is better for a retriever system. In this scenario, in the 2020 Guu et al. introduced REALM [15], a novel pre-trained BERT language model enhanced with external knowledge from a retriever. The paper shows how this improvement is mainly attributed to a pre-training task on a large text corpus, indicating that large language models could perform well on OpenQA. Language models have also demonstrated good performance in open-domain question answering by using stored information from pre-trained phase, without accessing any external knowledge or context [16]. However, they reveal that such language models can only perform as well as recent OpenQA systems if they are sufficiently large (over 11 billion parameters), making it difficult to constantly update them with new knowledge. Taking this into account, Izicard et al. in 2021 put forth a retrieval-reader approach [4] that utilizes a retriever similar to the previous DrQA[13], followed by a reader made up of a generative model T5 that is instruction-tuned on several tasks, including question answering. Recently, in 2022, an other OpenQA system has been proposed using a retriever based approach, followed by powerful generative language models such as BART [5], reaching the current state-of-the-art on OpenQA benchmarks 1.

3. Method

Our proposed OpenQA system involves a multi-stage retriever that uses an information retrieval system to collect a large corpus from open-source and internal documents. Then, in the second stage, an index is created to enable the efficient search of pertinent documents dividing each document into multiple text chunks, and then converting them into semantic vectors using modern Sentence Transformers models [17]. Each retrieved text is ultimately sent to a multi-stage reader that uses a fine-tuned Flan-T5 [12] instruction model to first get multiple articulated answers and, then, aggregating them.

1https://paperswithcode.com/sota/open-domain-question-answering-on-kilt-2
Figure 2: A full overview of the retriever system. The input document from the corpus is divided into several text chunks, which are subsequently transformed into vectors preserving their semantic meaning. When a question is presented, it is initially converted into a vector, followed by conducting a semantic search to identify the most similar text chunks. Together creating the final answer. The Figure 1 outlines our system break it down into two main components: the information retrieval system and the OpenQA system itself.

The information retrieval system gathers data from internal and external sources. We simply discuss how the system was developed since our work concentrates on the AI core. The information retriever is an external system containerized with Docker, accessible by FastAPI 2, and able to scrape various document kinds (pdf, html, word, etc.) and recover a full corpus of documents from the web. Specifically, we implemented different scrapers considering different sources (e.g., google.com) and using the Scrapy 3 framework. Using the information retriever for evaluation would have been problematic due to the constantly changing document corpus. To ensure a better evaluation of the question answering task, we kept the document corpus fixed, as explained in the Evaluation section.

The OpenQA system can handle natural language queries and provide answers based on a given corpus of text documents from the previous information retrieval system. In this paper, we will focus solely on this system, as it uses artificial intelligence models and can be compared to the "retriever-reader" architecture used by modern OpenQA systems. The specific retriever and reader implementations are detailed below.

3.1. Retriever

The retriever identifies relevant documents from a large corpus based on an input question. It’s main objective is to filters out irrelevant texts, as applying question-answering models directly to each input document would require excessive computation and ultimately decrease performance. Figure 2 illustrates the retriever component as divided into three main parts further detailed: text splitting, which divides long documents into smaller parts using a sentence-based approach; embedding, which uses an AI model to convert each text into a semantic vector; and top_k retrieval, which retrieves the top_k texts most similar to the input question based on the calculated similarity between the question and each document vector.

3.1.1. Text splitting

To create numerical vectors that represent text semantics, we used transformer models [14] based on Seo et al.’s approach [2]. Because these models have a maximum input length, longer texts must be divided into smaller parts, which can disrupt meaning and make analyzing long-term dependencies difficult. While the authors of the Dense Phrase Index divided each text into phrases [2], we grouped sentences until the maximum input length was exceeded. To split text into smaller parts for deep learning models, we use Python’s sentence-splitter library 4. Sentences are grouped together until the tokenized maximum input length is reached, creating one chunk. The second chunk keeps 50% of the sentences from the first chunk and adds more until the maximum input length is reached. This process is repeated for the whole document. Although it generates more chunks and redundancy, it helps preserve context.

3.1.2. Embedding

We use a pre-trained Sentence Transformer model [17] to encode text into a fixed-length vector. These models are based on a BERT Transformer [18] and fine-tuned for asymmetric semantic search. The msmarco-distilbert-base-v4 model by Hugging Face 5 has shown the best generalization ability, as it was trained on the msmarco corpus 6 that includes Bing questions and answers. We did not re-train a new model for a specific dataset in this first phase, as we aimed to obtain a generic system that performs well for different types of documents. Therefore, we used the sentence embedding model with frozen weights to encode each document and the input question.

3.1.3. Top_k retrieval

In order to find the most similar documents to a given query, a numerical vector from the query is created and then the cosine similarity metric is applied to identify the most similar documents from the corpus. To speed up

2https://fastapi.tiangolo.com/
3https://scrapy.org/
the search, the Faiss library \(^7\) from Meta is used, which is efficiently implemented in C and allows for faster retrieval of the top \(k\) most similar documents based on a cosine similarity metric. Once this step is completed, the semantic search is concluded, and the top \(k\) texts relevant to the specified question are returned.

### 3.2. Question answering

The system’s objective is to answer a specific question using a given list of input text. The answer can be located within one or multiple input texts or may not be present. The system generates an answer considering all the relevant texts retrieved, and if it is not present, will return an empty answer. Unfortunately, we cannot combine all the texts due to the model’s maximum input length constraint, and we cannot increase the maximum input length by opting for a bigger model, as this would sacrifice the system’s efficiency and require non-commodity hardware. Therefore, we consider each text as a separate input because the retriever’s previous splitting process ensured that the maximum length of each text was not exceeded. To perform the task, we chose a sequence-to-sequence model like Glass et al. in their work \([5]\). However, our choice fell on FLAN-T5-large \([12]\) released on Huggingface \(^8\) as it was fine-tuned on several instructions allowing to handle various natural language processing tasks and even tasks that it has yet to be specifically trained for. As shown in figure 3, the flan-t5 model is utilized twice with two distinct instructions: one for **generative question answering**, which extracts a detailed response from the given text, and the other for an **answer aggregation task**, which extracts a single final answer from the previous responses.

#### 3.2.1. Generative question answering

The model generates and explains an answer for each input text. In order to simplify the next step of aggregating different answers, it is essential to provide some reasoning in addition to the answer during this phase. After experimenting with various instructions, we identified a good prompt for the generation model: "Answer to the following question if it is possible, otherwise return None. $context:...$question:...Answer:...

At each step, the $context is replaced with text and the $question with the real question, resulting in a comprehensive input prompt for the language model. The generation was performed using a beam search decoding with 5 beams, a temperature setted to 0.4 and a fixed number of new tokens equals to 100 as we are interested in a meaningful sentence generation.

#### 3.2.2. Answer aggregation

This step will combine the previous answers into a single response. To do this, we will concatenate all the answers together to create a single input text for the model. Although the procedure is similar to the previous step, the instructions differ as we require a precise and singular response. In this case, we have identified a prompt for the generation model: "Answer with as few words as possible precisely to the question.$context:...$question:...answer:...

At each step, the $context is replaced with the concatenation of all the previous answers and the $question with the real question. The generation was performed using a greedy decoding as we are more interested in a small answer.

#### 3.2.3. Post-processing answer

During the system evaluation phase, a response is considered correct if it matches exactly one of the plausible ones. Therefore, this phase performs simple operations to clean up the response as much as possible: conversion to lowercase, removal of punctuation and white spaces.

### 4. Experiments

In this section, we report empirical evaluations of our open domain QA system. All the experiments reported concern only the Open domain QA system, and not the information retrieval system, which aims to generate a collection of textual documents from diverse sources, whether open or internal. As the resulting corpus varies depending on the specific use case (for internal purposes, internal documents supplemented with selected web news are preferred, whereas for general tasks, a

---

\(^7\)https://github.com/facebookresearch/faiss

\(^8\)https://huggingface.co/google/flan-t5-large
We consider the following datasets.

Table 1
The results of the OpenQA system. The Exact Match (EM) and micro-F1 scores are provided for each dataset, alongside the count of question-answer pairs utilized within each dataset.

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>EM</th>
<th>QA pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQUADv2</td>
<td>70.08</td>
<td>64.19</td>
<td>11,873</td>
</tr>
<tr>
<td>TriviaQA rc web dev</td>
<td>75.95</td>
<td>68.73</td>
<td>68,617</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>EM</th>
<th>QA pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-domain generative QA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TriviaQA unfiltered web dev</td>
<td>66.32</td>
<td>58.13</td>
<td>131,993</td>
</tr>
<tr>
<td>TriviaQA rc web dev</td>
<td>70.85</td>
<td>63.14</td>
<td>68,617</td>
</tr>
</tbody>
</table>

Reading comprehension QA: The system is required to extract the correct answer from an average of six input documents, which exclusively consist of evidence documents provided by TriviaQA for each question. Although the input corpus is smaller, this task is more challenging than a standard reading comprehension task, as there are no designated passages and the answer could potentially be in any of the input documents.

4.2. Evaluation

4.2.1. Reading comprehension QA

To evaluate machine reading comprehension, we used the SQUADv2 and rc TriviaQA datasets since they are commonly used for this purpose. We used the passages from the Squad dataset and the collection of evidence documents from TriviaQA for each question, so performance is mostly influenced by the question answering system. The table 4.1 shows our results for both datasets. Our system does not achieve state-of-the-art results on the SQUADv2 dataset, but this is not our main objective. We are not aiming to excel on a specific dataset or machine reading comprehension task where the passage is already provided, and the system must retrieve the answer. Consequently, models that are trained on SQUADv2 dataset for such a specific task tend to perform much better. On the other hand, the results related to the TriviaQA rc web dev dataset are comparable to those found in the state of the art, excluding those based on large language models (billions of parameters) as our model has 750M parameters. Nevertheless, a more careful and specific evaluation should be carried out for the other existing systems to assess whether they have been tested in the same scenario, especially in a zero-shot regime.

4.2.2. Open-domain QA

For this evaluation, we used the entire TriviaQA document corpus, consisting of approximately 487K documents. However, this fixed input corpus is not comparable to open-domain QA systems that typically use larger corpora, such as Wikipedia dumps [4]. The performance shown in the table 4.1 on the TriviaQA unfiltered web dev dataset is inferior to that of more modern open QA systems. This is probably due to the fact that in this initial evaluation of the system, we considered a fixed initial corpus, and the unfiltered TriviaQA dataset does not guarantee that the answer is present in the documents provided, so in many cases, the system may not be able to find the answer. Finally, the table 4.1 also reports the performance on the TriviaQA rc web dev dataset for this open domain scenario. These performances can be

---

9https://nlp.cs.washington.edu/triviaqa/

10https://paperswithcode.com/sota/question-answering-on-squad20

11https://paperswithcode.com/sota/question-answering-on-triviaqa

compared with the previous ones in the table 4.1, and as expected, the F1-score decreases by 5%, as the retriever is more deceived by the large number of documents in the corpus.

5. Conclusion

In this paper, we investigate a method to develop an openQA system, which relies on retrieving documents containing information relevant to the query and extracting the answer using an instruction fine-tuned language model, following a "zero-shot" approach. Despite its conceptual simplicity, we show that combining a "zero-shot" and a prompt engineering strategy is a practical and promising approach to building a flexible open-domain question-answering system. In future work, we would like to compare our whole system, including information retrieval, to the state-of-the-art.

References