Improving a Search Engine for Answering User Questions in Natural Language

Abdenour Chaoui

University of Paris Saclay

final year internship report

Improving a Search Engine for Answering User Questions in Natural Language

Author:
Abdenour CHAOUI

Supervisors:
Oana Balalau
Robin Reynaud
Pavel Soriano

2020/2021

Contents

List of Figures

Abstract

During this internship, we worked on improving an open domain question answering system. We addressed the document selection part which is structured as a text ranking task. The first step was to explore classical methods, also known as sparse retrievers, and to test those algorithms on our evaluation dataset. These methods produced only minor differences in performance. The next step was to employ deep language models, namely the BERT based architectures. A variety of techniques and designs were considered. First, we tackled the lack of data to train such models on the French language, followed by the definition of the problem (classification or ranking), and finally, we addressed the problem of limited text length in BERT-based models. The final results show a 12% improvement in performance over the original model.

Introduction

Humans established their first language roughly 50,000-150,000 years ago, and since then languages have continued to evolve and become more complex. From the inception of computers humans dreamt of giving them the ability to process, understand and communicate in human language. NLP or Natural language processing, a subfield of Artificial intelligence and linguistics, is the field working on achieving this goal. The availability of data, combined with the exponential growth of processing power,is driving the field to new heights and broadening our perspectives.
NLP has progressed to the point that it can be difficult to distinguish between a human and a machine in a discussion. NLP can be used in natural language comprehension, question answering and sentiment analysis.
Machine question answering is an important research area because machines can access and interpret massive pools of data in seconds, data that would take people examine manually their entire lifetime. This capacity can be very beneficial in a variety of ways, including automating processes for searching in textual data and extracting information that would have been difficult or impossible to uncover without these methods.
In this project, we will work on a question-answering system for French administrations in order to facilitate communication with their users and assist their employers. We'll look at the architecture of these systems, which are made up of two components: a document selection brick and an answer extraction brick. Our work will concentrate on the first component, and we will investigate several techniques to increase its performance, ranging from classical methods to more complex deep learning models.

Organization. This report is organized as follows:

- Section 1 contains information about the internship, the project context, and the internship's objectives.
- Section 2 presents a foundation on question answering and its modern architecture, then relates to the system currently utilized, its initial status, and the current dataset used for evaluation.
- Section 3 presents Classical approaches, together with their theory, implementation, and performance.
- Section 4 describes modern language models, their basic components, and a high-level overview of their design. This section concludes with the state of the art models presentation.

- Section 5 introduces the datasets used used to train neural models, together with some exploratory data analysis.
- Section 6 presents the project's methodology as well as the training procedures.
- Section 7 describes the experimental setup as well as the final results, followed by a discussion of these findings.
- The conclusion lays out a brief summary of what this report is about. In addition, it explores the challenges that we faced during implementation. It also explores contributions that could be made in future works.

1 Internship Presentation

This chapter outlines the motivations behind the project as well as the institutions involved in the development, followed by the objectives of the internship.

1.1 Problem formulation

French public administrations' online platforms are the citizen's access points to information and administrative procedures. These websites, such as service-public.fr, also allow the users to directly contact an agent if they are unable to find answers to their question. These direct inquiries cost time and resources for public officials and administrations.
Consequently, Etalab's LabIA, which is part of the DINUM (the Interministerial Digital Direction), is working on a project to provide an automated response service to users' questions. This project is called Piaf (Pour des intelligences artificielles francophones) initiative, which is also a collaboration with Inria.
This internship is part of the Inria Saclay CEDAR project team. This project-team is involved in the research area of highly scalable, parallel Big Data storage and processing tools, and paradigms of user interaction with Big Data.

1.2 Piaf project

Piaf is one of the projects of Etalab, a department of DINUM, that coordinates the design and implementation of the government's strategy in the field of data. Technically, the main objective of this project is to promote and develop French-speaking AI and the technologies that support it. It is set out in three stages. First, create an annotation platform based on a thorough scientific methodology inspired by the English-speaking equivalent SQuAD (Stanford Question Answering Dataset). Second, construct the dataset alongside a community of contributors including citizens, associations, researchers, and public officials. Finally, and third, once this dataset is built, test it and apply it to specific use cases experienced by administrations.
The first two steps have already been completed, and the project is progressing to the third and final step. Piaf question answering solution is one of the use cases under development, aiming to provide the community with an easily activated French question answering solution. Furthermore, it can be used as an interface intended for the administration agents that aims to make finding information easier, or via a chatbot-like interface that communicates directly with the user.
The present solution employs a two-step mechanism. First, the selection of a

subset of documents that will serve as contexts for the model to answer the question asked, then another step to extract the answer from these selected documents. The details of this approach will be discussed later in this report. Piaf findings demonstrate that the answer extraction step performs very well when given the proper context, but the document selection step often fails in this critical task.

1.3 Internship objectives

As part of their partnership, the internship is conducted between the Inria Saclay and the LabIA team. A bag-of-words scoring function or sparse retriever' BM25 is used in the present document selection paradigm. The project's objective is to work on this phase by examining various research approaches. First, we investigate the use of different sparse retrievers, next we investigate the use of deep learning models, particularly transformer-based language models, and their possible impact on the final results.

2 Background

From older designs and their components to the more contemporary retriever-reader architecture, this section gives a quick introduction of question answering systems. Then we investigate its constituents as well as their evaluation methods. Finally, we refer to the Piaf QA System and briefly explain the technologies and methods employed in its development.

2.1 Open Domain Question Answering

Question Answering (QA) aims to provide precise answers in response to the user's questions in natural language. QA can be divided into two types: Knowledge Base KB-QA and Textual QA. KB-QA searches for answers from a predefined knowledge base whereas Textual QA from unstructured text documents. Based on the available textual information textual QA is studied under two settings MRC (Machine Reading Comprehension) and Open-domain QA. To answer a question under MRC setting both the question and one context passage or document that contain the correct answer are provided, whereas the OpenQA setting tries to answer a specific question given a list of documents. In the second setting, in addition to finding the correct answer in a context passage, the model should first rank the documents by their likelihood of containing the correct answer.

2.1.1 Traditional Architectures

The traditional QA system usually comprises three stages. The first one is question analysis, used to facilitate relevant document retrieval,by generating search queries, and predicting the question type to facilitate answer extraction in the upcoming stages. The second stage is document retrieval where various models have been used among which are:

- the Boolean model is a simple model where all the present words in each document are recorded, and the queries are structured as Boolean expressions combined with Boolean operators of the form "word1 AND word2 NOT word3". Finally, the model returns 1 if the document fulfills the expression and 0 otherwise.

- Vector Space Models: first both queries and documents are transformed into vector representation then the similarity is computed using simple functions (e.g. cosine similarity)

- Probabilistic Models: Integrates the relationship between words into a model, for example, term frequency and document length.

- Language Model: for each document d in the collection a language model M_d such as ,the unigram language model, is constructed. Then for a query q the documents are ranked according to the probability p(M_d|q).

Finally, the answer extraction phase is responsible for extracting the answer from the context passage and relies on different matching methods for example word or phrase matching.

2.1.2 Modern OpenQA systems

Inspired by traditional architectures, Modern OpenQA systems are based on the "retriever-reader" architecture. The **retriever** can be thought of as an IR information retrieval system,it is designed to extract the relevant documents to a given question,it ranks the documents in the corpus according to their relevance to the question and returns the top k scoring documents, in other words retrievers are often formulated as a text ranking problem, which is the task of ranking documents in a corpus $C = \{d_i\}$ composed of an arbitrary number of textual documents, in terms of their relevance to a query q. While the **reader** is designed to infer the final answer from the received documents, it is commonly implemented as a neural MRC model. These are the two key elements of a modern OpenQA system.

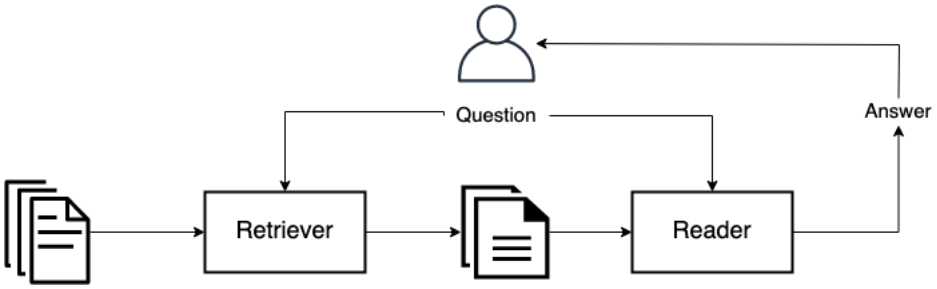

Figure 1: Modern OpenQA Architecture

2.1.3 Evaluation Metrics

In order to quantify the quality of the OpenQA system modules, several metrics are used. The evaluation of a model over a set of queries is called **a run**. Moreover, the evaluation can be performed on each module separately or as a pipeline configuration. Below, the most important metrics used in this project.

Retriever Metrics

The Retriever model receives in input a set of questions and a set of documents and gives in output, for each question a ranked list of documents that might

be useful for answering the question. In order to evaluate the retriever, we should have the gold standard, that is for each question, what is the correct document to retrieve.

Recall Recall is the fraction of correct retrievals from the queries in a single run. It is often evaluated at cutoff k,hence, Recall@k.

$$Recall = \frac{correct\ retrieval\ count}{Q}$$

Where, correct retrieval count, is the number of times the retriever was able to find at least one relevant document for a query in a singe run. And, Q, is the number of queries(questions) in a single run.The metric is simple to interpret but does not take into account the position of the relevant document also known as the rank.

Mean Average Precision (MAP), is the mean of average precision of queries. It is used when we have many relevant documents for each query.

$$MAP = \frac{1}{Query} \sum_{q=1}^{Q} AvePrecision(q)$$

AvePrecision is the mean of the precision scores after each relevant document is retrieved.

Mean Reciprocal Rank is the average of the reciprocal ranks of the first retrieved relevant document for a sample of queries Q, where reciprocal rank is the multiplicative inverse of the rank of the first correct answer.

$$MRR = \frac{1}{Q} \sum_{1}^{Q} \frac{1}{Rank_1}$$

Recall is considered to be a limiting factor for QA systems, and it is regarded as the system's upper limit. MAP and MRR, on the other hand, are measurements of retrieval quality and they provide an assessment of the rank.

Reader Metrics
The Reader model receives in input a set of questions and a context document for each question, and gives in output, for each question an answer in natural language. In order to evaluate the reader, we should have the gold standard, that is for each question, what is the correct answer to extract.

Exact match is a metric that checks if the predicted answers matches exactly the true answer for every query in a single run.

$$em = \frac{exact\ matches\ count}{correct\ retrievals}$$

correct retrievals is used when the reader is evaluated separately, i.e it is not penalized for the retriever errors. In case of the evaluation of the whole system the number of queries 'Q' is used.

F1 score definition is not straightforward in the case of natural language processing. The F1 metric measures the overlap between the true answer and the predicted answer, which makes it less strict than em. It is calculated the same way as in standard F1 score

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

TP : the number of tokens (usually characters in this context) present in both predicted answer and true answer.
FP : the number of tokens present in the predicted answer but not the true answer.
FN : the number of tokens present in the true answer but not in the predicted answer.

2.2 Initial status of the project Piaf

The system Piaf is based on Haystack, an end-to-end framework for building production-ready search pipelines for Question Answering and semantic document search.
This framework offers strong backends like Elasticsearch [1], FAISS [2] and transformers and it is highly customizable to integrate custom models. Piaf QA includes two building blocks, as described previously: a retriever for document selection and a reader for answer extraction. The reader is a transformer-based language model trained on a French QA dataset, And the Retriever is a TF-IDF like algorithm (BM25) provided by Elasticsearch backend.

To assess the system's performance, LabIA maintains datasets of their clients. **DILA** (Direction de l'information légale et administrative) that runs the service-public website is one of these clients. The DILA dataset is composed of

[1] www.elastic.co
[2] faiss.ai

3121 document from service-public.fr website and 380 user-asked queries and answers provided by public service agents. The dataset spans a wide range of themes such as social security, working conditions, legal procedures.The dataset is characterized by document that are considered long in the context of text ranking with an average document length of 913 word per document, and domain-specific terminology.
When supplied with an appropriate context, the reader is adequately efficient; nevertheless, the retriever brick frequently fails to offer the appropriate context in the full pipeline setup. In this project, we will focus on improving the retriever component of the Piaf system.

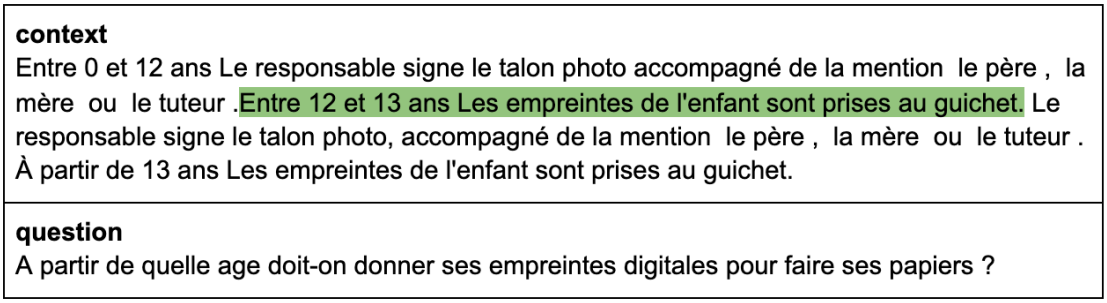

Figure 2: Example of a context with annotated answer and question in the passport file of DILA dataset

3 Sparse Methods

We now have a fundamental understanding of OpenQA technologies as well as the project's initial condition. The first stage of the internship is to investigate classical methods or what is known as sparse retrievers or searchers. The purpose of this section is to present these methods before implementing and testing their results on the DILA dataset.

3.1 Exact term matching

Exact term matching methods or sparse retrievers are based on classical IR methods. In exact-term matching the terms from documents and terms from a query have to match exactly to contribute to a ranking or relevance score. Usually, these terms are normalized to some extent for example by applying stemming. The similarity function between a document d and query q for these methods can be written as follows :

$$S(q,d) = \sum_{t \in q \cap d} f(t)$$

Where f is a function of a term and its associated statistics, the two most important of which are term frequency and document frequency. A central theme of early research on this topic was the exploration of various term weighting schemes for representing documents in vector space using easily computed statistics. Two of the most known of these methods are TF-IDF (term frequency–inverse document frequency) and BM25. These methods are still an entry point for many recent approaches in text ranking.

3.2 Theoretical background

Okapi BM25 is a probabilistic model used to estimate the relevance of documents to a given search query. It is an improvement upon the TF-IDF retrieval method.

$$Score(D,Q) = \sum_{i=1}^{n} IDF(q_i).\frac{f(q_i,D)(K_1+1)}{f(q_i,D)+k-1(1-b+b\frac{|D|}{avgdl})} \tag{1}$$

$$IDF(q_i) = \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1$$

where $f(q_i, D)$ is the term frequency in document D, avgdl is the average document length in the corpus. K_1 and b are free parameters.

LMDirichletSimilarity and **LMJelinekMercerSimilarity.** Unlike BM25, words in the query that are not present in the documents contribute to the score using smoothing techniques. LMDirichletSimilarity and LMJelinek-MercerSimilarity use smoothing techniques to estimate the relevance of documents to a given search query. - LMDirichletSimilarity

$$Score(D,Q) = \sum_{i=1}^{n} log(\frac{f(q_i,D) + \mu f(q_i,C)/|C|}{\mu + |d|}) \tag{2}$$

- LMJelinekMercerSimilarity

$$Score(D,Q) = \sum_{i=1}^{n} log((1-\lambda)\frac{f(q_i,D)}{|d|} + \lambda\frac{f(q_i,C)}{|C|}) \tag{3}$$

where $f(q_i, D)$ is the term frequency in the document D , $f(q_i, C)$ the term frequency in the collection C, $|d|$ is the total number of words in document $d, |C|$ is the total number of words in the collection C, μ and λ are free parameters.

Divergence from randomness (DFR). The DFR model assumes that the relevant words in a document are those whose frequencies diverge from the frequency indicated by a fundamental randomness model.

$$Score(D,Q) = \sum_{i=1}^{n} f(q_i,Q)w_{i,j} \tag{4}$$

$$w_{i,j} = [-logp(q_i|C)] * [1 - p(q_i|D)]$$

where $f(q_i, Q)$ is the term frequency in the query, $p(q_i|C)$ and $p(q_i|D)$ are the probability estimation of term in the collection and document respectively using randomness models such as Poisson, or Bose-Einstein.

Divergence from independence (DFI). The DFI model replaces the notion of randomness in DFR, with the notion of independence. A term in a document has more weight if its frequency diverges from its predicted frequency from the independence model.

$$Score(D,Q) = \sum_{i=1}^{n} f(q_i,Q)w_{i,j} \tag{5}$$

$$w_{i,j} = log_2(\frac{f(q_i,D) - e(q_i,D)}{\sqrt{e(q_i,D)}} + 1)$$

$$e(q_i, D) = f(q_i, C)\frac{|d|}{|C|}$$

$e(q_i|D)$ is the expected frequency. |d|,|C| are lengths of document and collection respectively in number of words.

3.3 Implementation

The initial step was to perform a sparse retrievers benchmark to utilize the findings as a reference point with the forthcoming parts of the project. The current system uses the default retriever BM25 with its default parameters. The goal is to explore the integration of other methods and search for parameters that produce the best results for the data.
Before proceeding to the retrievers, we first apply the necessary preprocessing to the data. In this part, we have performed the following preprocessing techniques.

- **elision** to remove elision from the beginning of a token e.g. (l'arbre -> arbre)
- **stemming** to reduce the word to its root form, this guarantees that diffrent variations of a word match.
- **stop words** to removes stop words from a token sequence e.g. (le,la,les)
- **lower case**

For each algorithm mentioned in 3.2. we perform the test with and without preprocessing and grid search of their parameters.
The test was performed using elasticsearch similarity modules, and piaf-ml testing framework on the DILA dataset.

3.4 Results

In our evaluation we use the metrics recall and MAP from 2.1.3.

	1	3	5	7	9	11	13	15
BM25	**0.71**	0.8	0.84	0.88	0.88	0.89	0.91	0.91
LMDirichlet	0.66	0.8	**0.86**	**0.89**	**0.92**	**0.92**	**0.94**	**0.94**
LMJelinekMercer	0.67	0.79	0.81	0.85	0.86	0.87	0.89	0.9
DFR	0.69	0.81	0.85	0.87	0.88	0.89	0.9	0.92
DFI	**0.71**	**0.82**	0.85	0.88	0.91	**0.92**	0.93	**0.94**

Table 1: Recall@k for different algorithms

	1	3	5	7	9	11	13	15
BM25	**0.71**	0.75	**0.76**	**0.77**	**0.77**	**0.77**	**0.77**	**0.77**
LMDirichlet	0.66	0.73	0.74	0.74	0.75	0.75	0.75	0.75
LMJelinekMercer	0.67	0.72	0.73	0.73	0.74	0.74	0.74	0.74
DFR	0.69	0.75	0.75	0.76	0.76	0.76	0.76	0.76
DFI	**0.71**	**0.76**	**0.76**	**0.77**	**0.77**	**0.77**	**0.77**	**0.77**

Table 2: MAP for different algorithms

The results presented in the tables represent the best grid parameter performance. Although some algorithms, such as LMDirichlet, surpass BM25 in terms of recall for high k, the results are close because of the nature of the algorithms(exact term matching) and BM25 is sufficiently performant.

4 Neural Language Models

Due to the success of deep learning in different tasks, various deep models on different NLP tasks have been created in recent years, greatly increasing efficiency. These specific-task models rely on pretrained word representations such as word2Vec or Glove as input which provides a remarkable semantic representation. Word representation models are typically trained on a large collection of textual data, then fed to the more complex specific-task models. However, each word is represented by a single prototype vector that does not vary with its context; as a result, polysemous words, or words that can have various meanings depending on their context, are impossible to distinguish.
Large-scale pre-trained language models that give contextual embeddings for words have recently substituted these models. Of particular interest are transformer networks, and especially BERT. The arrival of BERT has resulted in significant gains in performance across a broad variety of tasks. In the next part, we give a detailed overview of its architecture. We first describe the general structure of transformers and than we move to BERT.

4.1 Transformers

The improvements in general deep learning research may be credited for much of the recent improvements in the NLP field. Of particular interest are transformer networks. Compared to typical traditional models, transformers [Vaswani et al., 2017] provide several advantages such as effective modeling of long-term connections between tokens in a temporal sequence, as well as a more efficient training in general, by eliminating the sequential dependency on preceding tokens.
A transformer is an architecture for transforming one sequence into another. It adopts the encoder-decoder architecture. The full architecture of the transformer is shown in figure 13. A sequence is passed through a **tokenization** step to split it into smaller units called tokens (e.g. words, characters, subwords). The tokens are then fed to an **embedding layer**, which may be thought of as a look-up table to search for continuous vector representation (word embedding) $\psi(w_t)$ for each token in the sequence. Next, the **positional encoder** injects the positional information vector generated by the following equations:

$$PE_t = \begin{cases} PE_t(pos,k) = sin(\frac{pos}{10000^{2i/d_k}}) & \text{for k=2i} \\ PE_t(pos,k) = cos(\frac{pos}{10000^{2i/d_k}}) & \text{for k=2i+1} \end{cases} \tag{6}$$

where d_k is the dimensionality of the embedding vectors, k is the vector element position and pos is the position of the token in the sequence.

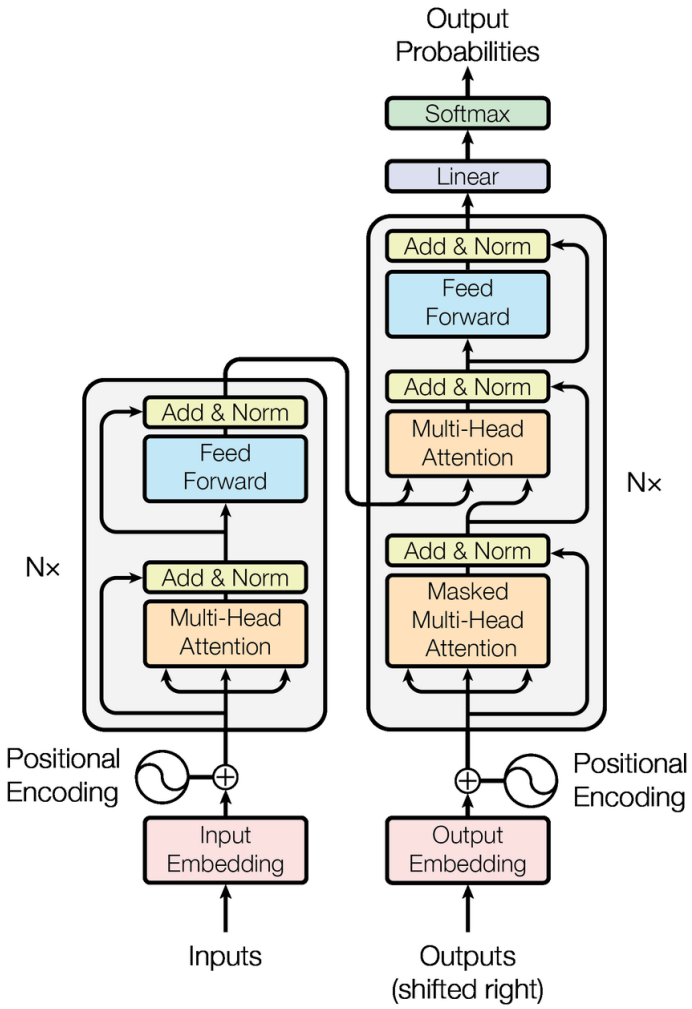

Figure 3: The Transformer model architecture [Vaswani et al., 2017]

For every word w_t in the sequence the positional embedding is added on top of the word embedding.

$$\psi^{'}(w_t) = \psi(w_t) + PE_t \tag{7}$$

4.1.1 The encoder layer

The encoder's role is to map the input sequence embeddings $\psi(w_t)^{'}$ for each token into another embedding vector that holds the learned information from the whole sequence. The encoder is broken into different parts. The **Multi-Head-Attention** is a module to calculate the sequence's words inter-dependencies or self-attention. Self-attention is calculated as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{8}$$

$$Q = \Psi W_Q \ , \ K = \Psi W_K \ , \ V = \Psi W_V$$

Where W_Q, W_K, W_V are learnable parameter matrices and $\Psi = [\psi^{'}(1), \psi^{'}(2), ..., \psi^{'}(T)]$ In multi-headed attention, the input vectors are projected to a smaller vector space, where each one is passed through separate self-attention with an independent set of K, Q, and V matrices, each one giving a different output called a head. The heads are then concatenated and transformed using a learnable parameter matrix W_0

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_N)W_0 \tag{9}$$

The output of the multi headed attention is added to the positional input embedding output Ψ through a **residual connection** then a **normalization step**, followed by **feed forward network** to project the attention to a higher dimension. Finally, there is another residual connection and normalization step. All of these operations are designed to encode the input to a continuous representation containing attention information.

4.1.2 The decoder layer

The decoder's role is to generate a sequence of words from a vector representation. It has the same basic components as the encoder topped with a linear classifier and softmax. First, a masked multi-headed attention, that works exactly like the encoder's multi-headed attention but since we are in a sequence to sequence task the input should only consider previously generated output tokens, so it employs a masking mechanism to disregard the unknown words. The second multi-headed attention matches the encoder and decoder inputs, it allows the decoder to select which encoder input should be prioritized. Finally, the next predicted word will be picked depending on the probabilities provided by the linear classifier and softmax block.

4.2 BERT : Bidirectional Encoder Representations from Transformers

BERT [Devlin et al., 2018] is a multi-layer bidirectional Transformer encoder pre-trained to create deep bidirectional representations. Based on the depth of the model, two versions of BERT were introduced, $BERT_{Base}$ and $BERT_{Large}$. It was trained on two tasks. The first is masked language model which masks a proportion of the input tokens at random and then predicts those masked tokens. The second is next sentence prediction, in which the model is given a pair of sentences and asked to guess if the second sentence is the subsequent sentence.

While BERT's main function is to transform a set of input tokens into a set of contextual embeddings, it can be fine-tuned to a variety of NLP tasks, just by adding an extra layer and fine tuning the architecture on the specific task inputs and outputs into BERT.

4.3 BERT for text ranking task

BERT can be applied to the task of two sentences classification (e.g. paraphrase detection). This setting can be used to finetune the model to perform a simple relevance classification between the query and the documents, that is to compute the probability $P(Relevant = 1|query, document_i)$. The documents in the corpus can then be ranked according to this probability. Although this technique seems promising, it requires computing the probability for the entire list of documents for each user query, and thus is impractical due to both the complexity of the model and the large number of documents. Another problem facing BERT-based models is that BERT was trained on 512 tokens inputs and it is not suitable for longer documents. In the next section, we discuss the different adopted solutions in text ranking literature.

4.4 State of the art

4.4.1 Multi stage re-rankers

In order to avoid running a costly neural network architecture on all pairs (query, document), the multi-stage re-ranker has been proposed in the literature (see Figure 4). This method is a two-stage approach. The first stage is responsible for candidate document selection using exact term matching-based methods, a very scalable method. The second stage re-ranks the candidate documents using the more powerful and computationally costly BERT-based model.

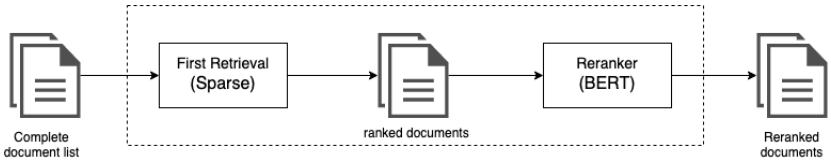

Figure 4: Retrieve then rerank

[Nogueira and Cho, 2019] proposed a model called **MonoBERT** for text ranking. It takes the sequence [[CLS] query [SEP] document [SEP]] as input, where [CLS] and [SEP] are BERT special tokens. Then passes the [CLS] token output embedding to a fully connected layer followed by a softmax activation function. The model was trained end-to-end using cross-entropy loss and

the scores are the probability of the relevant class of the softmax output. MonoBERT was only trained on shorter documents, hence, it did not address BERT size limit of 512.

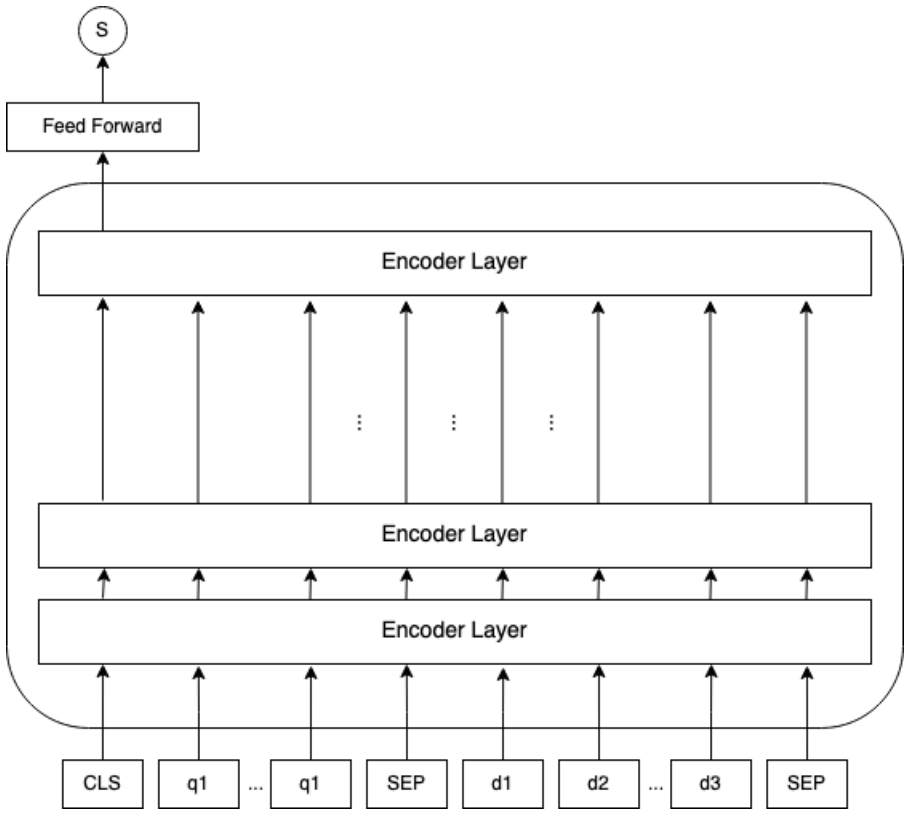

Figure 5: MonoBERT architecture: the input is passed with the form [[CLS] query [SEP] document [SEP]] through the BERT model. The CLS output embedding that summarizes the information of the whole input is then passed through a feed forward network to extract the score.

Document Ranking with Sentences (Birch) [Yilmaz et al., 2019] , uses the same architecture as MonoBERT, and is also trained exclusively on short documents. Birch, on the other hand, divides the documents into sentences and assesses the relevance score of each sentence. The total score for the document is the aggregation of these scores using the formula:

$$s_f = \alpha s_d + (1 - \alpha) \sum_{i=1}^{n} w_i s_i \tag{10}$$

where s_d is the first retrieval score, s_i is the i-th sentences score, α and w parameters are tuned manually.

Passage Score Aggregation [Dai and Callan, 2019]. During both training and inference, it segments documents into overlapping passages. During training, the passages are labeled following their original document (if the

document is relevant consider all the passages relevant and vice versa). During inference apply simple functions to aggregate the scores: max score, first passage score, or the average score.

Parade (Passage Representation Aggregation) [Li et al., 2020] adopts the same splitting method as the Passage Score Aggregation, taking overlapping segments for both training and inference. But it adopts a different aggregation strategy instead of aggregating scores, it aggregates the final [CLS] token embedding (average pool, max pool, softmax) then passes the new aggregated vector through a classification layer to get the relevance score of the full document.

4.4.2 Dense retrievers

The transition from exact matching techniques, to continuous dense representations that may capture semantic matches to improve model relevance is arguably the single most significant advantage brought about by deep learning approaches in text ranking. However, we have only studied models that utilize a sparse retriever as the initial step in their method, which may limit the overall performance. Meanwhile, several approaches, adopted a unique architecture, the bi-encoder, to overcome the problem of needing to traverse the whole list of documents for each user query.

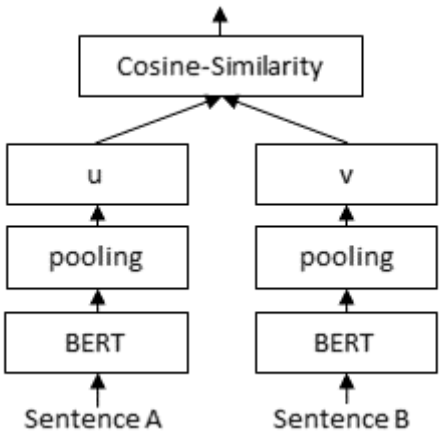

Figure 6: Bi-encoder Architecture

Bi-encoders use transformers to generate two separate representations for the document η_d and the query η_q, then calculates the similarity using a function ϕ.

$$P(Relevant = 1|q, d_i) = \phi(\eta_q, \eta_{d_i}) \tag{11}$$

The main advantage of this architecture is the creation of separate representations for queries and documents. After training is done, this allows to

pre-compute and save all the documents embeddings and during inference only compute the query embedding. The similarity can be the inner product of two vectors.

Sentence-BERT In the sentence-BERT model, siamese networks (networks that have the same weights while working on two different inputs) of BERT/RoBERTa are used to generate a separate representation for the documents and queries. The model is trained as a classifier.

$$s_i = cos(\eta_q, \eta_{d_i}) \tag{12}$$

Dense passage retriever (DPR) [Karpukhin et al., 2020] uses two separate networks to generate a separate representation for the documents and queries. The representation vectors are the [CLS] token from BERT and the similarity is the inner product between these vectors. The network is trained to maximize the distance between negative and positive passege embeddings.

Representation-focused BERT (RepBERT) [Zhan et al., 2020] Uses a Siamese BERT network and average pooling of their output to generate a separate representation of the documents and queries. It uses a pairwise loss function of one positive and one negative passage score to optimize the loss function.

$$\ell = 1 - (s_d^+ - s_d^-) \tag{13}$$

5 Datasets

Training Neural Language Models necessitates massive datasets; the current la DILA dataset is clearly insufficient for this task, This is why we search for more suitable datasets. Text ranking datasets in English, such as MS Marco [Bajaj et al., 2016], are very large allowing a wide range of liguistic variety. However, because we are working for French administrations, such datasets in French do not exist, therefore we rely on french QA datasets, which provide queries, and a unique context document for each query. In this section, we will present the datasets that were used in the project, as well as the strategy that was employed to make them compatible with the task at hand. Finally, we examine the data to have better knowledge and an overview of the model design process.

5.1 French QA datasets

Deep language models require a large training dataset to be effective, this is why we looked for all the available French QA datasets. It's worth noting that all the dataset we are going to present follow the Squad (Stanford Question Answering Dataset) [Rajpurkar et al., 2016] style. Squad consists of questions on Wikipedia articles, with each question's answer being a text segment or span.

SquadFR is the first attempt to create a french QA dataset, it was created by automatically translating SQuAD v1.1 dataset using Google Translate API and adapting the answers span to the French context. The dataset contains more than 100K question answer pairs.

FQuAD [d'Hoffschmidt et al., 2020] Unlike SquadFR, FQuAD is a French Native Reading Comprehension dataset of questions created by higher education students on a set of Wikipedia articles. The dataset contains more than 25K question answers pairs.

Piaf To attain the goal of establishing an open QA dataset, Piaf dataset was created using the same protocol as Squad v1.1. The questions were created by a community of contributors. The dataset has huge growth potential as it is an ongoing project funded by the French government. The dataset contains 9K pairs of questions and answers.

5.2 Text ranking dataset creation

For the text ranking task, we need a dataset that contains a set of ordered documents for each query. The available French QA datasets provide for each query only one context document that contains the answer. Inspired

from the training schemes adopted in the DPR (Dense Passage Retriever) [Karpukhin et al., 2020] let the desired dataset format be of the form:

$$D = \{q_i, \text{positive context} : [d_i^+], \text{negative contexts} : [d_{i,1}^-, \ldots d_{i,n}^-]\}_{i=1}^m \quad (14)$$

Where m is the number of queries, the positive context contains the explicitly available documents associated with the query in the original QA dataset, whereas the negative contexts (documents that do not contain the answer) are generated following three strategies:

- **Random.** Select any random passage of other queries in the dataset.
- **Gold.** During training, select the contexts of other queries in the batch.
- **BM25.** Select the top passages returned by the classical method BM25 documents that don't contain the answer but match most question tokens.

In this project, the third strategy of selecting the top passages from BM25 is used to generate the negative contexts.
We sum up the process: first, we combine the three French QA datasets, SqadFR, FQuAD, and Piaf and we remove duplicate questions. Then for each query in the resulted combination, we save the associated context as positive context, and the 10 highest-ranking documents selected by BM25 in relevance to the question as negative context.

5.3 Dataset Analysis

In this part, we explore the generated dataset after combining the three French QA datasets and applying the negative context generation approach.
The merged dataset constains 125440 unique questions. Table 3 indicates the frequency of the various types of questions found in the dataset. The table highlights the fact that questions staring with **que** account for nearly half of the questions; this can be explained by the fact that SquadFR the automatically translated dataset represents nearly 100k questions, and the question formulation **que** includes both the English **what** and **which**.

Question	Frequency [%]	Example
Que	49.7	Quel niveau de mobilité économique ...
Qui	16.7	Qui a enseigné les enfants ...
Combien	6.9	Combien de temps dure ...
Quand	6.4	Quand la Vaisesika darsana ...
Où	4.1	Où se trouve le Museo ...
Comment	3.2	Comment les lois de New York ...
Pourquoi	1.5	Pourquoi le zinc était - il ...
quoi	4.0	De quoi est composé ...
Others	7.5	Le président de la Chambre ...

Table 3: Question type by frequency

The dataset also counts 28144 unique documents. Each passage can serve as positive context for several queries and can also appear as a negative context for other queries. But one document can not be both a positive and a negative context for the same query. Finally, there are 1375690 distinct question-documents in the dataset, regardless of their relation (positive, negative).

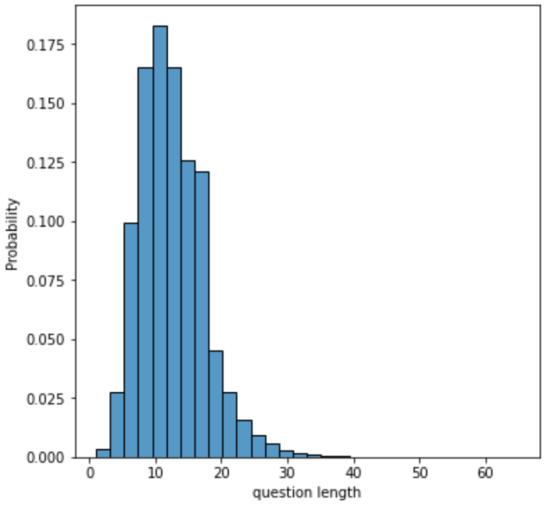

Figure 7: Histogram of question lengths (number of tokens)

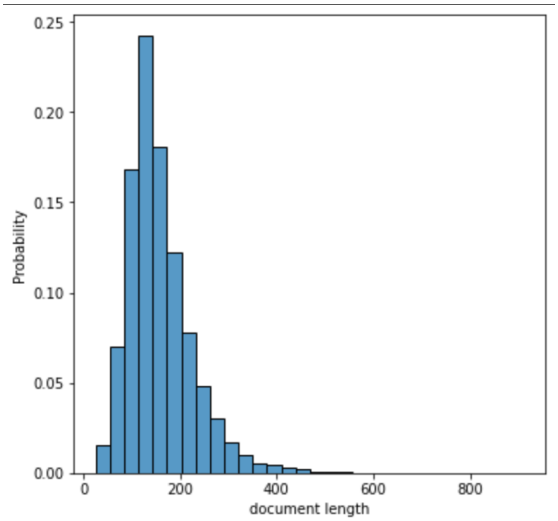

Figure 8: Histogram of documents lengths (number of tokens)

The number of tokens in a question ranges from 1 to 65, while for documents from 25 to 912. Figures 7 and 8 show that most questions have less than 30 tokens and most documents have less than 400 tokens. As a result, most question-document combinations fall within the 512 token limit of BERT.
The sum of the query and document tokens, as well as the three special tokens, should not exceed 512 in the case of the cross encoder technique when the input has the form [CLS] query [SEP] document [SEP]. To this end, we are

going to check the inputs that surpass that threshold to decide the training strategy.
Table 4 shows that 99.8% of the inputs are in the range [0,512] in compliance with BERT size requirement.

Number of tokens	[0,112[	[112,212[	[212,312[	[312,412[	[412,512]	+512
Frequency	206829	856903	254176	44382	11138	2262
Cumulative relative frequency	0.150	0.773	0.958	0.990	0.998	1.000

Table 4: Input length distribution

Figure 9 depicts the data imbalance between the two classes. This is expected because of the nature of the creation of the dataset and should be considered during the training phase.

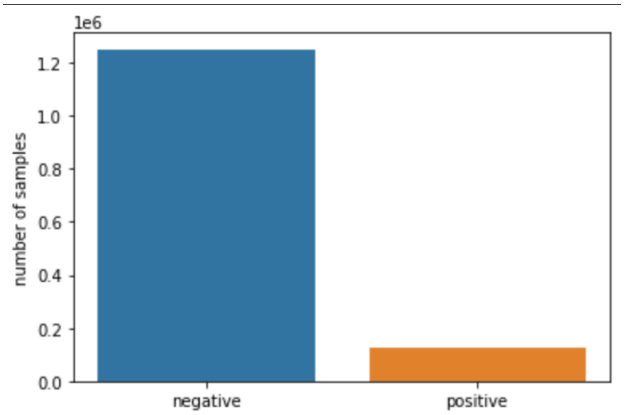

Figure 9: Data distribution per class

5.4 Data split

We split the data into a train set, validation set and a test representing $80\%, 10\%, 10\%$ respectively. We make sure that no question is repeated in one set or between sets.

	Train set	Validation set	Test set
# unique questions	100352	12544	12544
# question-document pairs	1098746	137344	137513

6 Methodology and implementation

In this section, we propose a model to improve the QA system's text ranking module. For this, we use a BERT-based re-ranker model.
In the first phase we use BERT embeddings directly without finetuning to save time and resources, while experimenting with different methodologies and architectures. In the second step we finetune the entire model once a viable method is identified.

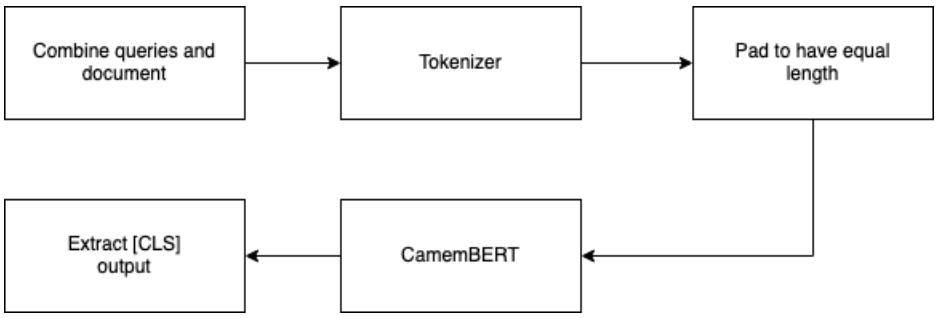

Figure 10: The Transformer model architecture

6.1 BERT based model

To develop a re-rank approach, discussed in the previous section, the MonoBERT model is utilized as a starting point and then modified to our dataset and, of course, the French language.
So far we have only discussed BERT, a language model completely trained on English language data. But there exists another variation of BERT released by the Google team, mBERT or (multilingual BERT) which is a model based on BERT architecture but trained on 104 languages. After the success of BERT and its variant mBERT a lot of research focused on this area and tried to create monolingual models based on their architecture.
CamemBERT Camembert is a pre-trained language model that is completely trained on French language data and outperforms multilingual models in several tasks. CamemBERT is a multi-layer bidirectional Transformer based on RoBERTa which itself is an improved version of BERT. From this point on we rely on CamemBERT as a base model in our architecture.

6.2 Embeddings extraction

Most of the approaches that uses BERT as a base model for text ranking task rely solely on the CLS final output embedding, which encodes the entire

input. Due to the voluminous size of data and the huge number of BERT-based models parameters, we first save the embeddings, use them as input our specific task models, and do not finetune the whole model.
To extract the CLS token output embeddings no text preprocessing is required, we only have to combine the query and documents from the dataset to form the required input ([CLS] query [SEP] document [SEP]) . This input is then passed through a special Camembert tokenizer. The tokenizer splits the input sequence into tokens and assigns a unique ID to each token. For each sequence the unique ID list is passed through CamemBERT, so we finally extract the [CLS] token representation or T_{CLS}.
The extraction of all CLS representations T_{CLS}, takes almost 9 hours to generate vectors of size 768 for all the question-document pairs in the datasets.

6.3 Classification Approach

The concept behind this approach is to train the model as a binary relevance classifier to provide scores of relevance to a particular input. The contextual representation of the [CLS] token (T_{CLS}) is passed into a multi-layer neural network to calculate the probability s_i that the candidate d_i is relevant to a question q. We generate a s_i score for each candidate individually.

$$s_i = sigmoid(WT_{[CLS]} + b) \tag{15}$$

Where W is weight matrix, and $T_{[CLS]}$ is the CLS output embedding.
We train the model for re-ranking using binary cross-entropy loss:

$$\ell = y_i log(s_i) + (1 - y_i) log(1 - s_i) \tag{16}$$

where y_i is the input label and s_i is the score of candidate documents.
Before training, we address the huge imbalance in the data between the two classes. To this end, we assign weights to the loss computed for different samples differently. The weights for each class are computed as follows

$$w_{class} = \frac{n_{samples}}{n_{classes} \times n_{class}} \tag{17}$$

Where $n_{samples}$ is the total number of samples, $n_{classes}$ is the number of classes, n_{class} is the number of samples in one class.
The model is trained using Adam optimizer and we use grid search tuning for the following parameters

- **learning rate** [0.01 , 0.005 , 0.001 , 0.0005]
- **Hidden layer size** [256 , 512 , 1024]
- **Number of layers** [1,2,3]

- **Dropout** [0.2 , 0.3 , 0.4]

The validation dataset was utilized for early stopping when the validation loss did not improve for 10 epochs, as well as hyperparameter tuning when we select the best model on the validation dataset. The selected model has 0.0005 learning rate, 1024 hidden size with 3 hidden layers and 0.3 dropout. In the next part, the model is referenced as $MonoBERT_{classifier}$.

6.4 Pairwise Approach

Inspired form the pairwise loss calculation method used in RepBERT [Zhan et al., 2020] that uses bi encoders. We apply the same loss but using cross encoder architecture. The goal of this approach is to train the model to enforce a margin between two documents with different labels. The approach is more suitable for ranking tasks. During training the contextual representation of the [CLS] token (T_{CLS}) of two inputs of the same query but different documents with opposite labels (relevant/not-relevant) are passed through a multi-layer neural network to calculate their respective scores s_i and s_j. Then a pairwise loss function or margin ranking loss is used

$$\ell = max(0, y_i(s_i - s_j)) \tag{18}$$

In this approach, the problem of imbalanced data is no longer an issue, as we pair each positive context with all the negative context associated with the same query.
The model is then trained under the same settings as the classification approach and is referenced as $MonoBERT_{ranker}$.

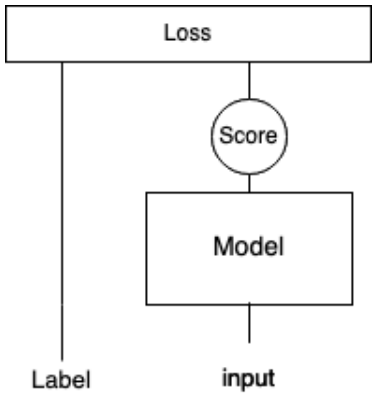

Figure 11: classification approach loss

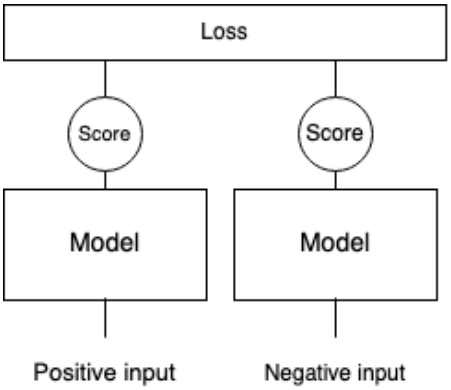

Figure 12: ranking approach loss

6.5 Complementary scoring

Sparse retrievals already achieve decent results so the idea is to combine their results with the results of a continuous representation model. The weakness of one model is the strength of the other: one matches precise tokens but cannot manage vocabulary mismatches; whereas the other allows semantic matching but loses granular information. The two types of models can complement one other, hence we use a linear combination of their scores.

$$s_i(q, d_i) = \alpha s_{Sparse}(q, d_i) + \beta s_{re\_ranker}(q, d_i) \tag{19}$$

Given that Sparse scores, such as BM25, have no upper bound, employing this approach requires scaling these scores across documents in one query before conducting any sort of combination. To scale document ratings, we utilize the Min-Max method.

$$s_{Scaled}(q, d_i) = \frac{s_{Sparse}(q, d_i) - min_{d_j \in D}(s_{sparse}(Q, d_j))}{max_{d_j \in D}(s_{sparse}(q, d_j)) - min_{d_j \in D}(s_{sparse}(q, d_j))} \tag{20}$$

The selection of *alpha* and *beta* is accomplished by performing a grid search with values ranging from 0 to 1, with a 0.1 increase on both alpha and beta.

6.6 Fine-tuning the complete model

After the training of classifier and ranker approaches, and exploring the complementary scoring, the final step is to train the complete architecture and finetune Camemebert on the downstream task of ranking documents.
The model was trained to minimize the margin ranking loss, as there is no imbalance problem and it is more suitable to the data and the task. Since the transformer is already a complex model, we only use one hidden layer of size and an output layer followed by a sigmoid. We used Adam optimizer with a learning rate of $1 \times 10 - 5$, with a batch size of 12 , and 0.3 dropout.
The finetuning was performed on an Nvidia Tesla V100 GPU. As recommended by BERT authors the model was trained for 3 epochs each epoch runs for approximately 16 hours.

6.7 Long Documents Ranking

Up to now, due to the nature of the dataset 99% of documents have length less than 512 tokens, we have only considered short document, but in the real system this situation should be considered. One example is the DILA datasets where 62% of the data is longer than 512 tokens.

For this end, we employ the following splitting strategy. Each document is divided into fixed-length passages with a percentage of overlap between successive passages.
Finally, we aggregate the scores to generate one score for each long document

$$s_f = \alpha s_{BM25} + \beta \frac{\sum_i^p s_i}{p} \tag{21}$$

where p denotes the number of passages, sf denotes the final document score, and si denotes the individual passage score.

7 Results and discussion

7.1 Experimental setup

We test the models using the dataset presented in Section 5. The test set contains queries that have not been seen by the model yet. For each query, the top 10 documents returned by the BM25 retriever are passed to the model. The model generates new scores for these documents and reranks them.

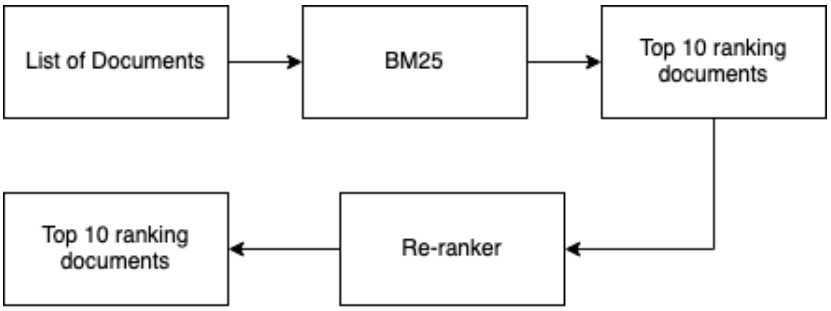

Figure 13: The Transformer model architecture

We do a second experiments using the DILA dataset, which contains long documents. The setup is the same as the top 10 documents returned from BM25 are passed to the model. The difference is that in long documents we adapt the strategy in 6.7. Each of these documents is divided into fixed-length passages 380 token with 120 token overlap.(Our experiments shows that this setting gives the best result)

7.2 Evaluation Metrics

We use two metrics from retriever metrics section 2.1.3 , MAP and two version of Recall@k. Recall@10 is used to demonstrate the upper bound of re-ranker models. If the relevant document is not provided in the list of 10 documents, the re-ranker will not be able to identify it, no matter how performant it is. MAP metric is used to asses the overall ranking improvement. Finally Recall@1 to measure the improvement of getting the relevant document on the top position.

7.3 Results

The results of the models on the test set are presented on table 5. The first row is the baseline model BM25, followed by classifier approach, the re-ranker approach then then their variant complementary. Finally the best performing model $MonoBERT_{Ranker}(Finetuned)$.

	Recall@10 [%]	Recall@1 [%]	MAP [%]
BM25	85	63	71
$MonoBERT_{Classifier}$	85	31	48
$MonoBERT_{Ranker}$	85	44	58
BM25 + $MonoBERT_{Classifier}$ (Complementary)	85	65	72
BM25 + $MonoBERT_{Ranker}$ (Complementary)	85	67	74
$MonoBERT_{Ranker}(Finetuned)$	85	**75**	**80**

Table 5: Result table of the different approaches on test set of (SquadFR+Fquad+piaf) dataset

	Recall@10	Recall@1	MAP
BM25	89	71	77
BM25 + $MonoBERT_{Ranker}(Finetuned)$ (Complementary)	89	**77**	**80**

Table 6: Result table of on the DILA dataset

7.4 Discussion

According to the results in 5, BM25 is already highly powerful. A brief examination at the correctly detected documents reveals that they share the same keywords and even phrasing as their corresponding queries. This is explained by the technique in which these datasets are annotated. A document is provided to the annotator, and then the annotator should formulate questions that can be answered by that document. Hence, the annotators tend to use the same keywords and phrasing as the documents.
$MonoBERT_{classifier}$ was not unable to outperform BM25 in terms of both recall@1 and MAP. It achieved limited results, with nearly half the performance of BM25.
Despite that $MonoBERT_{Ranker}$ outperformed $MonoBERT_{classifier}$ by 13% in terms of recall@1 and 10% in terms of MAP. However, it is still outmatched by

BM25. These findings demonstrate that the ranking approach is more suited to the task and data than the categorization approach.
The semantic matching power of $MonoBERT_{Ranker}$ and $MonoBERT_{classifier}$ can still be considered because they succeed in retrieving documents where BM25 fails, therefore a suitable technique to merge them with BM25 is justified.
The BM25+$MonoBERT_{classifier}$ and BM25+$MonoBERT_{ranker}$ use the complementary approach 6.5, to exploit both lexical and semantical representation slightly improves the results in both Recall@1 and MAP compared to BM25.
The final model $MonoBERT_{Ranker}$ as expected gave a huge leap of improvement in terms of both Recall@1 and MAP. This enhancement can be attributed to train the encoders section to separate the embeddings prior to the feed forward or classification part.
We visualize the embeddings generated by CamemBERT by class before and after fine-tuning to ranking class using t-SNE (t-Distributed Stochastic Neighbor Embedding) [van der Maaten and Hinton, 2008] which is a very powerful technique for visualizing multidimensional data. Figures 14 and 15 shows that fine-tuning gives the auxiliary model much more meaningful input to generate adequate scores.

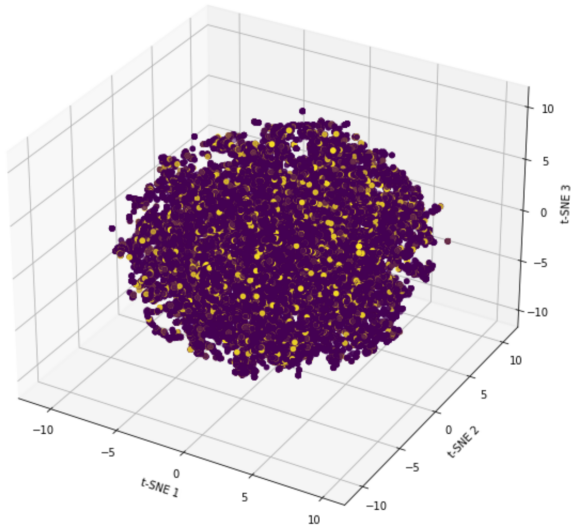

Figure 14: 3 dimensions representation using t-SNE of embeddings by class before fine-tuning, yellow=relevant

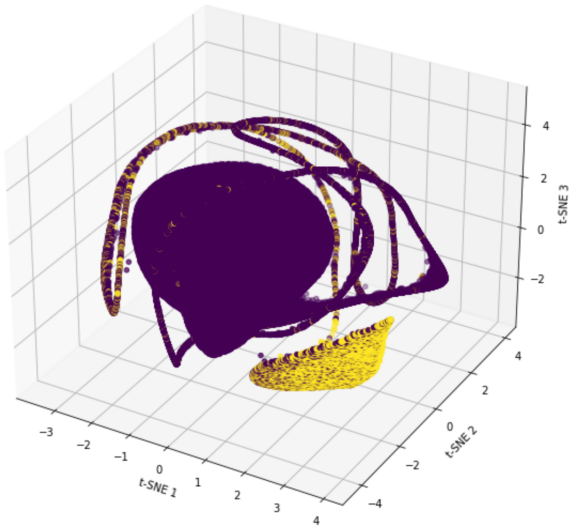

Figure 15: 3 dimensions representation using t-SNE of embeddings by class after fine-tuning, yellow=relevant

Using the Long document ranking method in 6.7. The proposed model improves the results for DILA dataset in terms of both recall@1 and MAP, according to the results in 6. The improvement is not as significant as in

the test set due to two factors: first, the small number of questions in the DILA dataset, and second, the splitting strategy, which may result in some contextual information being lost in the document.

Conclusion

In this project, we tried to explore the use of contextual deep learning models as text rankers, to improve a document selection module of a question answering system. The starting point was the exploration of classical methods and comparison of their performance. After introducing the basics of neural language models, we searched for an adequate dataset to train these models, which yielded the fusion of three French QA datasets. Finally, we employed a text ranking solution based on the CamemBERT model. We have tried different approaches inspired by the state of the art to finally reach satisfying results, improving the model's performance by 12% in terms of recall@1 and 8% in terms of MAP.

Future work. The study completed yields promising results for the text rating task using contextual deep learning models. The approach's disadvantage is its reliance on a first searcher or sparse retriever, which acts as a bottleneck for the performance. Hence, the next step is to explore the bi-encoder architecture to try to improve the current results.
Another observation made during development is the specificity of the data, which contains a lot of legal and administrative terms. The model does not understand certain words, such as "pacs" (Pacte civil de solidarité) and declinations of it. Hence training camemBERT from its initial checkpoint on the domain specific documents using the masked language model, where 15% of the input is masked and the model tries to predict the masked words, will give the model a better understanding of the data and allows a better comprehension of the specific domain terms and contexts.

Bibliography

[Bajaj et al., 2016] Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., and Wang, T. (2016). Ms marco: A human generated machine reading comprehension dataset.

[Dai and Callan, 2019] Dai, Z. and Callan, J. (2019). Deeper text understanding for IR with contextual neural language modeling.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

[d'Hoffschmidt et al., 2020] d'Hoffschmidt, M., Belblidia, W., Brendlé, T., Heinrich, Q., and Vidal, M. (2020). Fquad: French question answering dataset.

[Karpukhin et al., 2020] Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and tau Yih, W. (2020). Dense passage retrieval for open-domain question answering.

[Li et al., 2020] Li, C., Yates, A., MacAvaney, S., He, B., and Sun, Y. (2020). Parade: Passage representation aggregation for document reranking.

[Nogueira and Cho, 2019] Nogueira, R. and Cho, K. (2019). Passage reranking with bert.

[Rajpurkar et al., 2016] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100, 000+ questions for machine comprehension of text.

[van der Maaten and Hinton, 2008] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

[Yilmaz et al., 2019] Yilmaz, Z. A., Yang, W., Zhang, H., and Lin, J. (2019). Cross-domain modeling of sentence-level evidence for document retrieval.

[Zhan et al., 2020] Zhan, J., Mao, J., Liu, Y., Zhang, M., and Ma, S. (2020). Repbert: Contextualized text embeddings for first-stage retrieval.