Diploma Internship
Report

Keyword Extraction - Research and Implementation

Submitted by

Phuoc Khanh LE
University of technology in Compiègne

Under the guidance of

Boris Flûckier

Sinequa SAS
Paris

07/02/2022 - 22/07/2022

Remerciements

First, I would like to thank the University of Technology of Compiègne and the Machine Learning Team at the Research and Development (R&D) team at Sinequa for the instructive six months of my internship.

During the duration of my internship, in addition to the technical issues regarding the internship topics, there were also non-technical issues. Thus, I would like to thank Jordane Lamarche for helping me on the administration side. I also thank the IT team at Sinequa for the technical support of my workstation.

I would like to express my deep gratitude to Loïc Dagnas, Basile Van Cooten, Steffen Kirres, Rémi Claeys, Youva Vanlaer, Arnaud Duvieusart for teaching me many things and giving me enthusiastic advice, as well as for sharing their precious time.

Finally, I would like to express my great thanks to my internship tutor, Boris Flükiger, for his lessons and materials, for the time he shared with me, and for sharing a lot of experience and knowledge with me as for having inspiring me by his hardworking spirit.

Contents

Abstract

Deep learning is becoming increasingly important in business, especially in the technology company in Natural Language Processing (NLP) and Computer Vision. **Sinequa** is a pioneer company in applying deep learning to resolve NLP problems. As their product aims to solve various situations in processing text and documents, Sinequa Research and Development (R&D) team is trying their best to explore and integrate the deep learning solution into their application. Hence, as a machine learning engineer intern in the R&D team, the project's objective is to research and integration the *Keywords extraction* problems.

Keywords Extraction is extracting relevant chunks of words from a document to best capture and represent its content. Having keywords helps the reader get the document's gist at a glance and browse quickly through many documents. Hence, it is precious for the user experience of search engines to enhance previews. In conclusion, this is a worthy feature to explore, research and integrate if possible.

The research on this problem has come a long way since my internship. The previous research has found various datasets on the target problems. Based on these datasets, they have evaluated different methods: unsupervised learning and supervised learning. In their conclusion, *unsupervised learning* has the lovely property of not depending on the training data distribution, but it has a seriously worse performance compared to *supervised learning*. The supervised approach can teach a model to extract keyphrases by minimizing some loss function that, in turn, maximizes the F1 score on exact matches. This method works well with transformer-based models such as BERT. This is the main track of my internship for research.

After achieving a good performance in research, I integrated the Keywords extraction feature into Sinequa's product. This is the first try to see if a keyword extraction model can perform well in the context of production. The feature is implemented in two primary interfaces of the application: *http serving routes* and *Command line for runnable model*. These two functionalities worked well at the end of my internship.

In my resume, during my six-month internship, I ported the research in applying keyword extraction in solving keyword extraction problems and integrated the solution into Sinequa's software. My internship report traces the different stages of the realization of the projects as well as the needs, the constraints, the difficulties encountered, and the solutions selected to set up the solution.

1 Introduction

1.1 Company presentation

Sinequa[1] is a software company specializing in enterprise search. They develop a single but complete product, the Sinequa software[3], end-to-end software to master information within a company. The software links to all relevant sources of information within the company, indexes properly all documents (in particular unstructured documents such as PDF files), and is then able to retrieve them to better answer a query. The heart of the software is the search engine, with many imbued in Natural Language Processing. Moreover, Sinequa also builds a whole platform with a user and admin interface to create a highly customizable product. The admin can easily create Search Based Application (SBA)s to fit the client's needs, for example, to find all relevant information on a customer and display it in a dashboard, or recognize entities within a text.

Sinequa has been named a leader several consecutive years in the famous **Gartner magic quadrant for insight engines**[4] (figure 1) and in the similar **Forrester Wave: Cognitive Search Q3 2021**, which is reputed to have a great impact on business decisions. Those benchmarks are very popular in the United States and will certainly help Sinequa's reputation overseas.

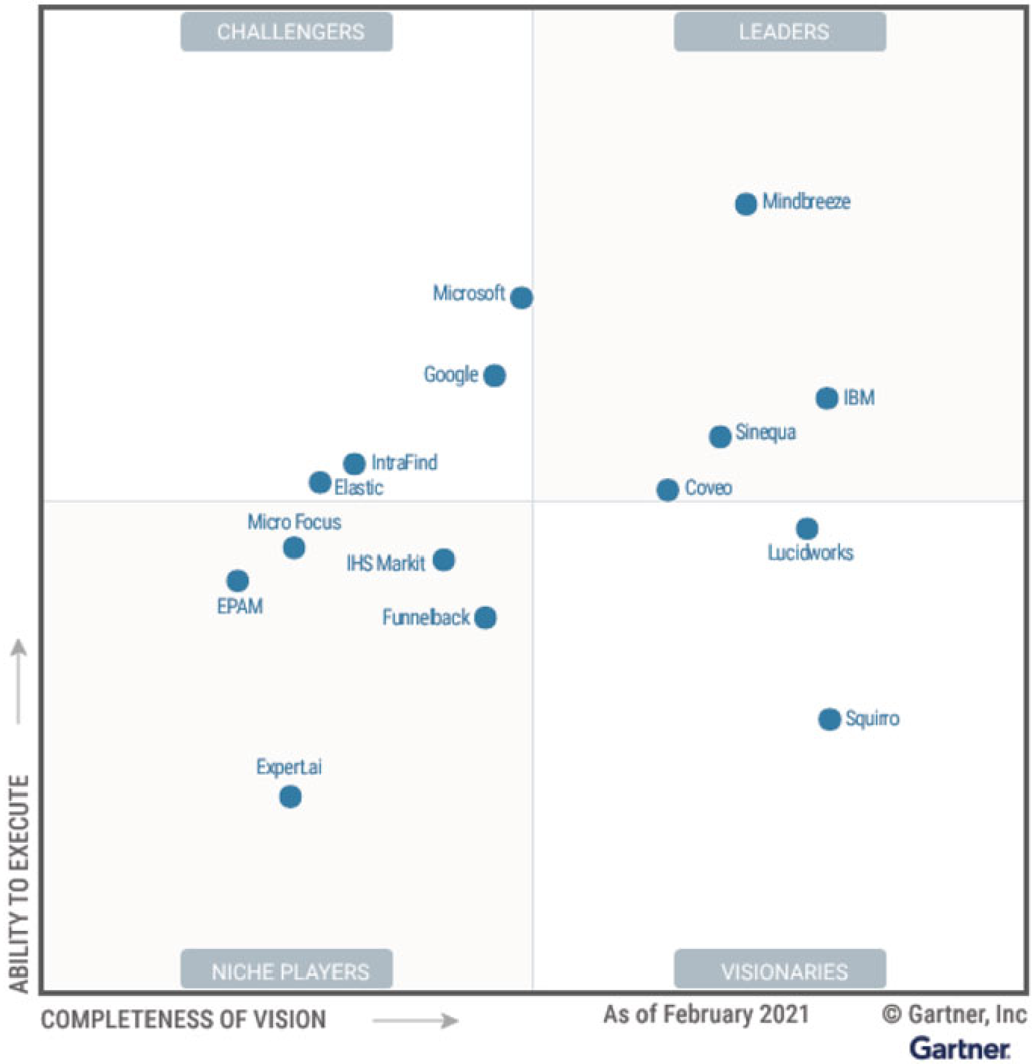

Figure 1: Gartner magic quadrant for insight engines.

Having prestigious clients like NASA, Pfizer,...[5], is very useful for one's reputation. It helps potential clients make a business decision with confidence that other companies would approve it, making it much easier to justify the said decision to investors and shareholders.

Sinequa usually works with consulting cabinets whose work is to implement Sinequa properly in the client company's existing software and intranet. The long-term endorsement of Sinequa relies on the technical abilities of the consultants. For this reason, Sinequa is highly interested in helping them perform efficiently, for example, with custom development. For example, Sinequa writes custom connectors to link the client's data lakes to the Sinequa indexer and database. This is why they have over 200 different kinds of connectors to this date. Figure 3 briefly explains how the Sinequa application works. The figure 2 is an example of the interface when making a search.

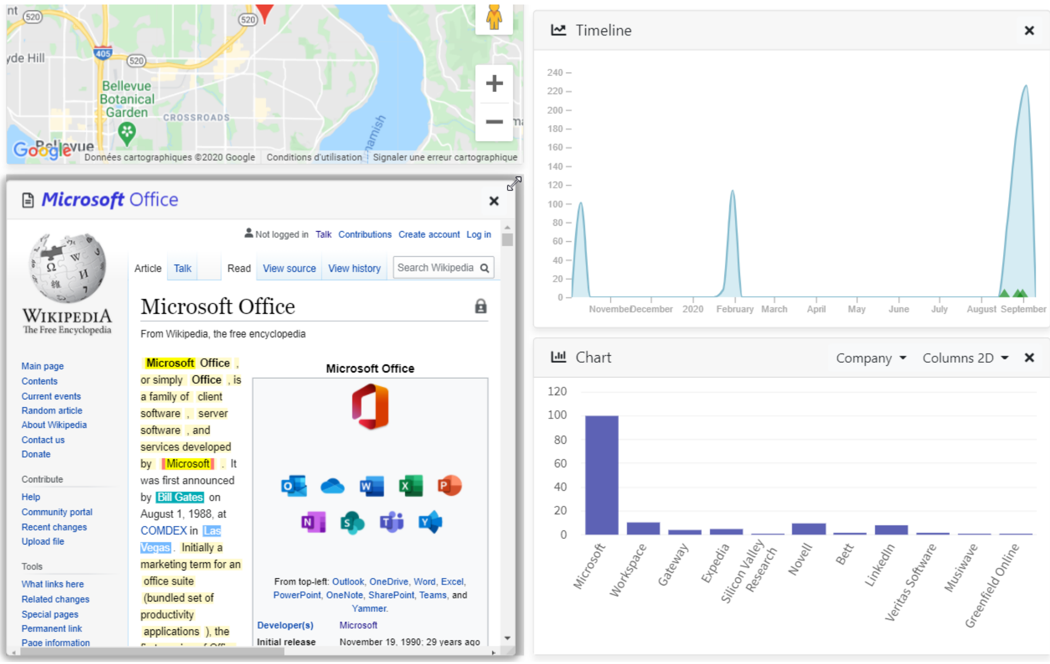

Figure 2: Example of Sinequa application's interface.

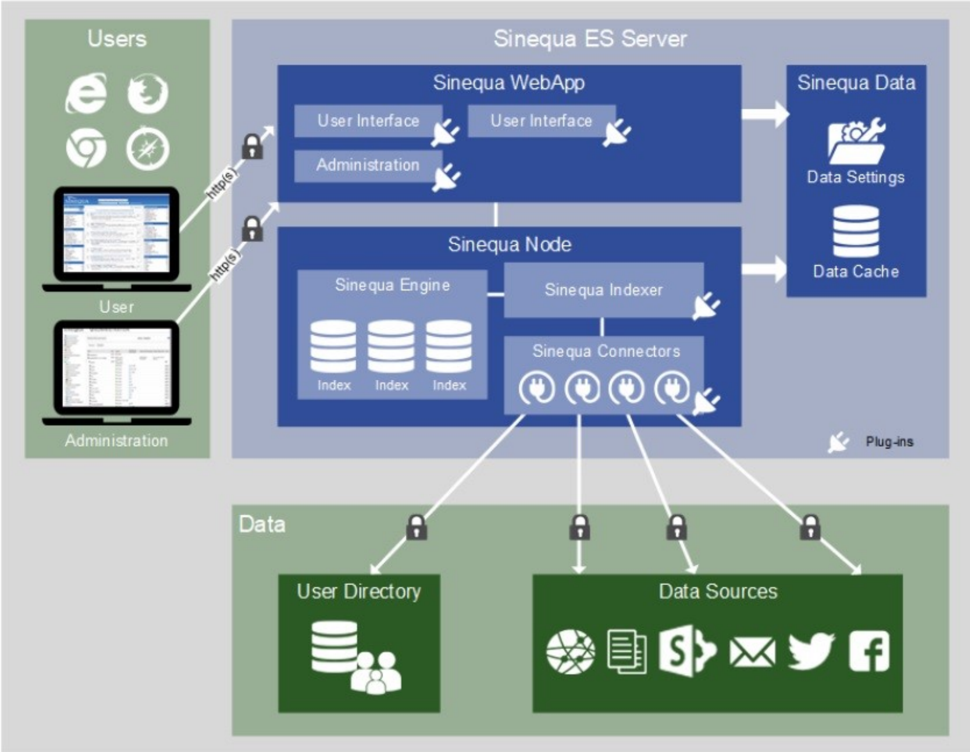

Figure 3: Scheme of Sinequa application. Data from various sources is collected by the *connectors*. Then, the data is indexed document by document. The indexed documents are stored in the database. The application can perform search through this database.

1.2 Project presentation

The main functionality of Sinequa software is a Search Engine, which depends on the documents' capacity to understand. A new project was initiated at Sinequa to enrich the information from the document. Its name is Text Augmentation. The primary purpose is to augment or add as much information as possible (like key-words, topic,...) for each indexed document. My project is the first component of Text Augmentation, named Keyword Extraction.

Keyword extraction is a text analysis technique that automatically extracts the most used and most essential words and expressions from a text. It helps summarize the content of texts and recognize the main topics discussed. Through many years, applying deep learning has achieved state-of-the-art (SOTA) in this task. The internship aims to deliver a model of deep learning that performs a good result and apply the model in Sinequa's production. Hence, my internship has two main parts: research and implementation.

VECTORIZATION OF TEXT USING DATA MINING METHODS

In the **text mining** tasks, textual representation should be not only efficient but also interpretable, as this enables an understanding of the operational logic underlying the **data mining** models. Traditional **text vectorization** methods such as TF-IDF and bag-of-words are effective and characterized by intuitive interpretability, but suffer from the «curse of dimensionality», and they are unable to capture the meanings of words. On the other hand, modern distributed methods effectively capture the hidden **semantics**, but they are computationally intensive, time-consuming, and uninterpretable. This article proposes a new text vectorization method called Bag of weighted **Concepts** BoWC that presents a document according to the concepts' information it contains. The proposed method creates concepts by **clustering** word vectors (i.e. word embedding) then uses the frequencies of these concept clusters to represent document vectors. To enrich the resulted document representation, a new modified weighting function is proposed for weighting concepts based on statistics extracted from word embedding information. The generated vectors are characterized by interpretability, low dimensionality, high accuracy, and low computational costs when used in **data mining** tasks. The proposed method has been tested on five different benchmark datasets in two data mining tasks; document **clustering** and **classification**, and compared with several baselines, including Bag-of-words, TF-IDF, Averaged GloVe, Bag-of-Concepts, and VLAC. The results indicate that BoWC outperforms most baselines and gives 7% better accuracy on average.

Figure 4: An example of keywords extraction.

The same research task had already been explored in a previous internship. They have explored various paths to resolve the problem: Unsupervised or supervised, Keyword Extraction[6] or Keyword Generation[7]. Their research helps us to start in a good direction and reduce our time on another path. Following the conclusion of the previous internship, we continue the research on maximizing the performance in the production constraint. In addition, applying the latest technology like Bert[8] refrains us from applying on very long documents. Hence, our research extends to applying the model to long documents. The expected result is a deep learning model checkpoint that can perform well on various domains and situations of documents.

The second part is to implement the feature of Keyword Extraction, using the result of the research part, into the product of Sinequa. As explained above, the Sinequa product is an end-to-end software to manage all client company documents. Hence, it is expected to perform as much as possible the NLP tasks. The Keywords Extraction feature is planned to be implemented at the end of my internship. My work in this part is to implement the data pipeline, from the original text to keywords, into the product of Sinequa. I am following the production's constraints, and performing the same result as the research part is the main challenge of my second part. The expected result is the C# code that can use a model checkpoint and perform keyword extraction inside Sinequa's software.

2 Research on Keyword Extraction

2.1 Data selection and analyze

In a deep learning project, data is one of the essential features. Data is used for training and evaluating the algorithm. Hence, choosing and processing the data is extremely important. At first, we choose which datasets to train and evaluate. There are some significant constraints for the dataset to train. The training data must be **freely available for commercial** because Sinequa's product is used in business. The freeness of the data must be cited in the license of its paper or the GitHub containing the data. Secondly, we do not know the topic on that the algorithm will be used. Hence, training data should be variations on the topic. After considering the recommendation of the previous internship and the research in the depth of each dataset's paper, we decide to choose these four datasets :

- Inspec[9] : A dataset of abstracts of scientific journal papers from Computer Science collected between the years 1998 and 2002.
- Semeval17[10] : A dataset of paragraphs selected from ScienceDirect journal articles, evenly distributed among the domains of Computer Science, Material Sciences and Physic.
- Pubmed[11] : A dataset of full-text papers collected from PubMed Central, which comprises over 26 million citations for biomedical literature from *Medline*, life science journals, and online books.
- Kptimes[12] : A Large-Scale dataset of news documents paired with editor-curated keyphrases.

After obtaining the data, analyzing the data is an important step. Although there are many properties to analyze, we choose to present only the most important ones. These are the properties to analyze :

- Number of documents.
- Length of document.
- Number of keywords for each document.
- Number of words in each keyword (length of keyword).

After analyzing all the datasets, we obtain the result in the table 5. The result is collected from the paper of the dataset.

Dataset	# of doc	Avr length of doc	Avr # of keywords	Avr length of keyword
Inspec	2000	162	6.365	2.705
Semeval17	500	492	14.8	2.2
Pubmed	500	3992	5.835	1.4
Kptimes	260.000	921	2.7	1.5

Figure 5: Data analyze result.

Considering the result, we conclude :

- We have the document in a large range of length of document (from 162 words to 3992 words).
- Kptimes has too much more documents than other datasets.
- There is no relation between length of document and number of keywords in a document. The average number of keywords depends on the way of annotating the data.
- The average length of keyword is around 2.

2.2 Data pipeline

A data pipeline is a list of processing steps to transform raw data into a format that can be used for training and evaluation. As each dataset comes from different sources and formats, it is not easy to apply the model simultaneously to different ones. Hence, we decide on a common pivot format for all datasets. As raw data contains too much unnecessary information, we extract only the document as *text* and *keyphrases* as label. The *url* is used as the document's id. We saved the pivot format as a JSON file containing a list of samples. Although the pivot format possesses information needed for keyword extraction, it is impossible to use this data for training. At first, the algorithm applies only to a specific format. Hence, the pivot format must be preprocessed. Secondly, we have to retrieve the position of all keywords as the algorithm uses this information for training. This is the transformation from pivot format to preprocessed format. The data pipeline is described in the figure 6.

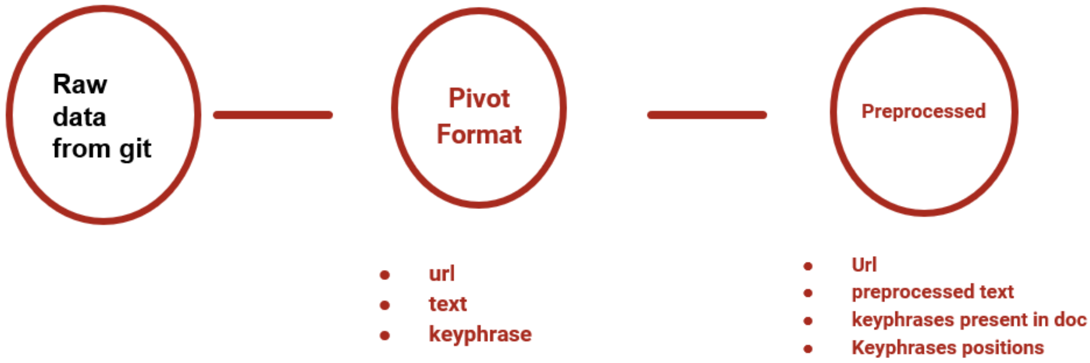

Figure 6: Data pipline for each dataset

The data pipeline is implemented by a python script for each dataset from raw to pivot format and one familiar python script from pivot to preprocessed format. After applying these scripts, we obtain the format that can be used to train and evaluate the algorithm.

2.3 Description of Bert-JointKPE and reproduce the paper

Open-domain KeyPhrase Extraction (KPE) aims to extract keyphrases from documents without domain or quality restrictions, e.g., web pages with variant domains and qualities. Recently, neural methods have shown promising results in many KPE tasks due to their powerful capacity for modeling contextual semantics of the given documents. However, most neural methods currently prefer to extract keyphrases with good phraseness, such as short and entity-style n-grams, instead of globally informative keyphrases from open-domain documents. Bert-JointKPE[13] proposes the architecture to resolve the problem by capturing both local phraseness and global informativeness when extracting keyphrases. For this section, we would like to introduce the term intuitive algorithm and explain our implementation in-depth.

2.3.1 Algorithm description

Architecture. Here is BERT-JointKPE's framework: first of all, the text is passed into a BERT[8] model to extract one token embedding per token. Bert[8] is a pre-trained language model of google. It can embed each document token into a numeric vector containing abstract information in the current context. At Sinequa, they are called Deep Language Model. The token embeddings are then fed into a custom CNN with various windows size (from 0 to the maximum length of a keyphrase), creating an N-gram representation for every possible N-gram. Those N-gram representations are then fed to the chunking network, which will select which N-grams are keyphrases or "Chunks" thanks to a linear binary classification layer.

Those chunks are fed to the ranking network which will rank them according to their salience and assign a score to each of them with a linear layer. The training is done jointly on the combined loss of the chunking and ranking network: $L = L_{Chunk} + L_{Rank}$. For the L_{Chunk}, the cross-entropy is computed for every chunk where the chunk label is 1 if the chunk represents a keyphrase and 0 otherwise. For L_{Chunk}, the hinge loss in pairwise learning to rank is computed on exact matches with ground truth by minimizing the score of non-keyphrases minus the score of real keyphrases. The figure 7 visualizes, in brief, the architecture of joint Bert KPE.

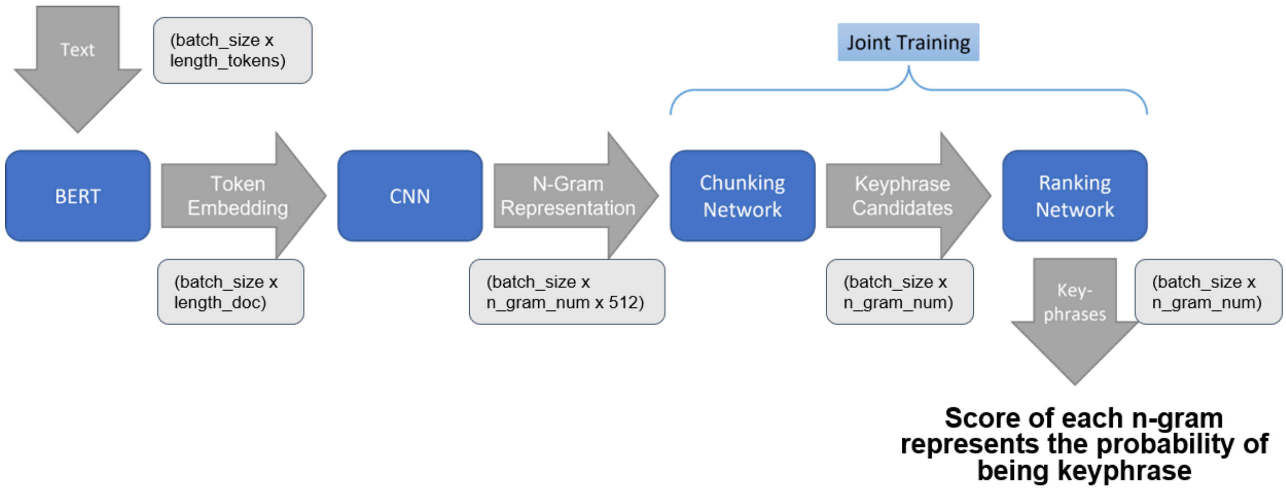

Figure 7: Bert Joint KPE architecture.

Input and output. The algorithm demands a specific structure of input be trained and evaluated. The input includes: tokenization of the document, position of all ngrams and position of all different ngrams.

Advantage and disadvantage. JointKPE have its own advantages and disadvantages.

+ **Advantages:**

  – At first, by using pre-trained deep language model, the algorithm have ability to understand the context of each word and also the influence of each token on general text.

  – Secondly, the model can extract long keywords which usually explain better the topic of the document.

  – By the report of the paper, the model can adapt on various topics : medicine, science, news, ...

+ **Disadvantages:**

– The model can not indicate which phrase is a keyword document, but attach a score for each phrase. The higher the score, the more probable that the phrase is a keyword.

– Using Bert[8] base model limit the length of sequence that can process by the algorithm. The maximum length for pre-trained model of Bert is 512 tokens. Hence we can not apply on long document.

Metric. Based on the context of the algorithm, we decide the metric to evaluate the model. Although we consider keyword extraction a classification problem, there is no F1 score because the model does not indicate precisely how many keywords are for a document. Hence, we use a variant of the F1 score, which is the F1 score at k as k is the top k candidates with the highest scores. After extracting top k phrases, we calculate the F1 score by the formula :

Precision = $\frac{TruePositive}{TruePositive+FalsePositive}$

Recall = $\frac{TruePositive}{TruePositive+FalseNegative}$

F1 = $\frac{2*Precision*Recall}{Precision+Recall}$

We note that **F1@k** is the F1 score with k candidates. In our research, we often use F1 score with 3,5, and 10 candidates, hence, we note them F1@3, F1@5, F1@10.

As true positive is the number of common keyword between candidate and list of keyword in label, false negative is the number of keyword that is predicted as not keyword, false positive is the number of candidates that are not the keywords.

For example, a document has 7 labeled keywords: deep learning, supervised learning, data, data mining, deep language model, natural language processing. The model proposes 5 candidates: deep learning, supervised learning, language, data, processing. There are 3 matches: deep learning, supervised learning and data. Hence, the precision is $\frac{3}{5} = 0.6$, recall = $\frac{3}{7} = 0.42$, the F1@3 score is 0.494.

2.3.2 Re-implementation using PyTorch Lightning

Although the algorithm's author offers the code on their GitHub, we re-implement the model in a new framework of Pytorch Lightning[16]. First, currently, Sinequa tends to have a common framework for all projects. It is helpful for the team to understand, debug, and reuse the code of a new project or repo, which is our case. The framework used at Sinequa is Pytorch Lighting. Secondly, having our implementation helps us understand the algorithm better. Hence, we can find out at which point we can change to improve the model. Finally, Pytorch Lightning is a novel framework that provides various conveniences: easy implementation, management of computational resources (GPU[14], TPU[15]), and adaptation to the code of Pytorch. In conclusion, the algorithm is re-implemented with PyTorch Lightning.

PyTorch Lightning is the deep learning framework with "batteries included" for professional AI researchers and machine learning engineers who need maximal flexibility while super-charging performance at scale. Lightning organizes PyTorch code to remove boilerplate and unlock scalability. While PyTorch is straightforward to use to build complex AI models. However, once the research gets complicated and things like multi-GPU training, 16-bit precision, and TPU training get mixed in, the implementation becomes difficult and introduces many bugs. PyTorch Lightning solves this problem precisely. Lightning structures PyTorch code so it can abstract the details of training. This makes AI research scalable and fast to iterate on. In addition, it is easy to implement Pytorch Lightning. Most of the function is already implemented under the hood. Hence, we just need to re-write the core of the model.

These are main parts of Joint-KPE to implement in Pytorch Lightning :

- The first part is different components of the model. Based on the figure 7, there are 3 main block : deep language model like bert, a convolutional network , and 2 fully-connected dense.

- The second part is the interaction between the blocks of the model. In fact, this is the calculation from input to the output of the model.

To verify the exactness of implementation, we run the code on the same configuration as the author's code with the same dataset for training and evaluation. The result is showed in 8. As expected, our implementation achieves the same result reported by the papers.

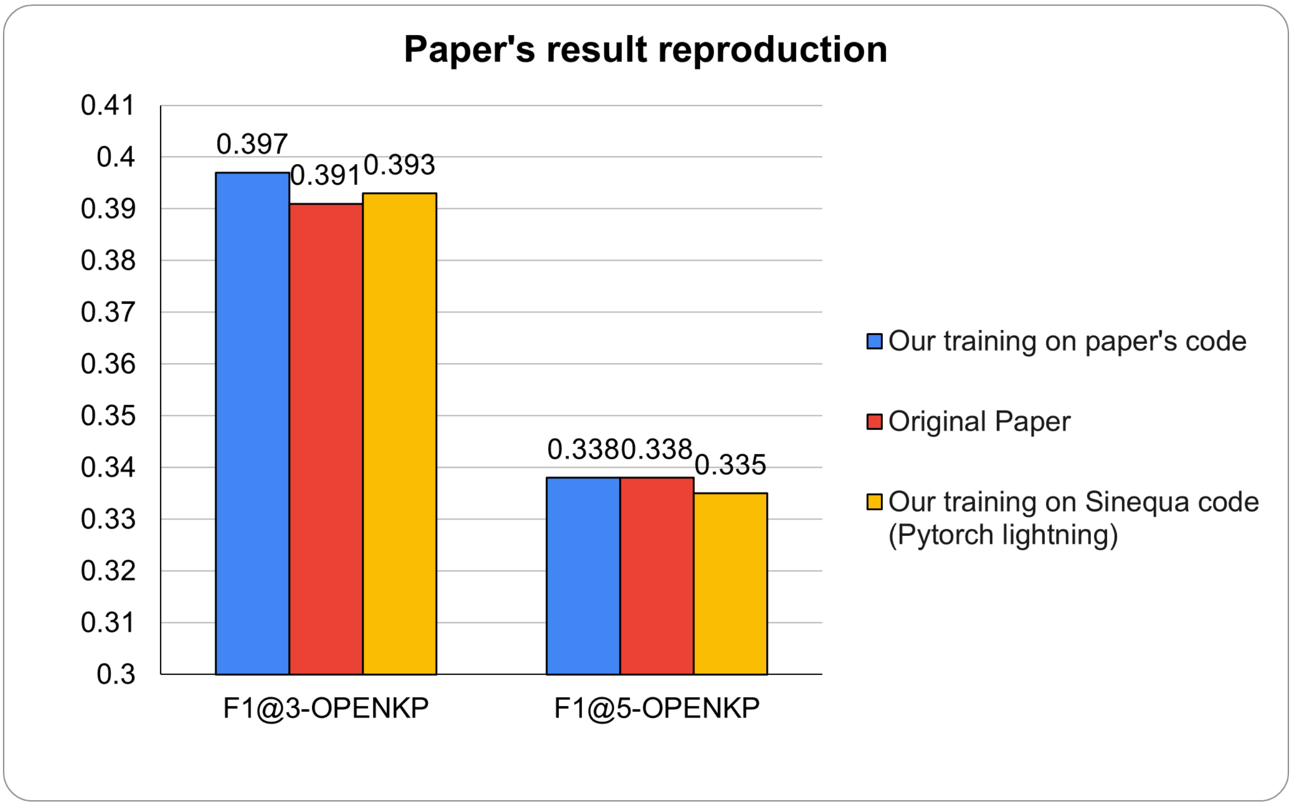

Figure 8: Paper's result reproduction.

2.4 Improvement of model's performance

After having the same result reported by joint-KPE's paper, we try to improve its performance. This is the highest priority task in my internship. We apply a common strategy to accomplish this task :

- Read the paper or the code to find the point to improve.
- Have an assumption for an idea that may improve the model.
- Have a metric that can prove the effectiveness of the model.
- Implement the metric and the idea and verify if the idea is a good track to follow.

Not all of our ideas are effective, and the report does not describe all of them.

2.4.1 Combining datasets

To benefit from the advantage of having datasets from various sources, concatenating all of them is our first idea. As explained in the prevision section, the datasets come from different sources and topics. Hence, combining them creates the most generalized dataset that we have.

Evaluation. At first. evaluating a generalized dataset makes us observe the model's capacity to adapt to different topics and situations. There are two ways of evaluating :

- *Zeros-shot evaluation on each component dataset*: observe the model's performance on a different topic.
- *Evaluation on the combination of test sets of all datasets*: observe the model's performance in a general situation.

Training. Secondly, the model will be more generalized when training on a generalized dataset. When faced with different types of distributions, the model has to learn beyond recognizing patterns in a single document; it has to learn the general principle of "what is a good keyphrase." Indeed, varying the type and source of training data reduces the confusion factors. For example, if all documents are scientific papers, the model will learn that rather complicated words are usually significant and people's names are insignificant. However, in a news article, it may be that a person's name is indeed essential. When learning about both types of data, the model must acknowledge that not all documents are identical. Hence, when faced with a new type of document, it will not assume it is exactly like the training distribution; therefore, the zero-shot performance is expected to rise.

Result. To prove the idea's effectiveness, we train the model on each dataset and evaluate it on another dataset. Then, we trained one model on the combined dataset and evaluated it similarly. The image 9 present the result on first situation. It is reasonable that model training on each dataset performs well on its own test set but badly performs on different tests. The image 10 shows the performance of model training by combined dataset. It proves that the model can perform well on all four datasets. On dataset Kptimes, the model even performs better than Training on kptimes. This phenomenon is called meta-learning[25].

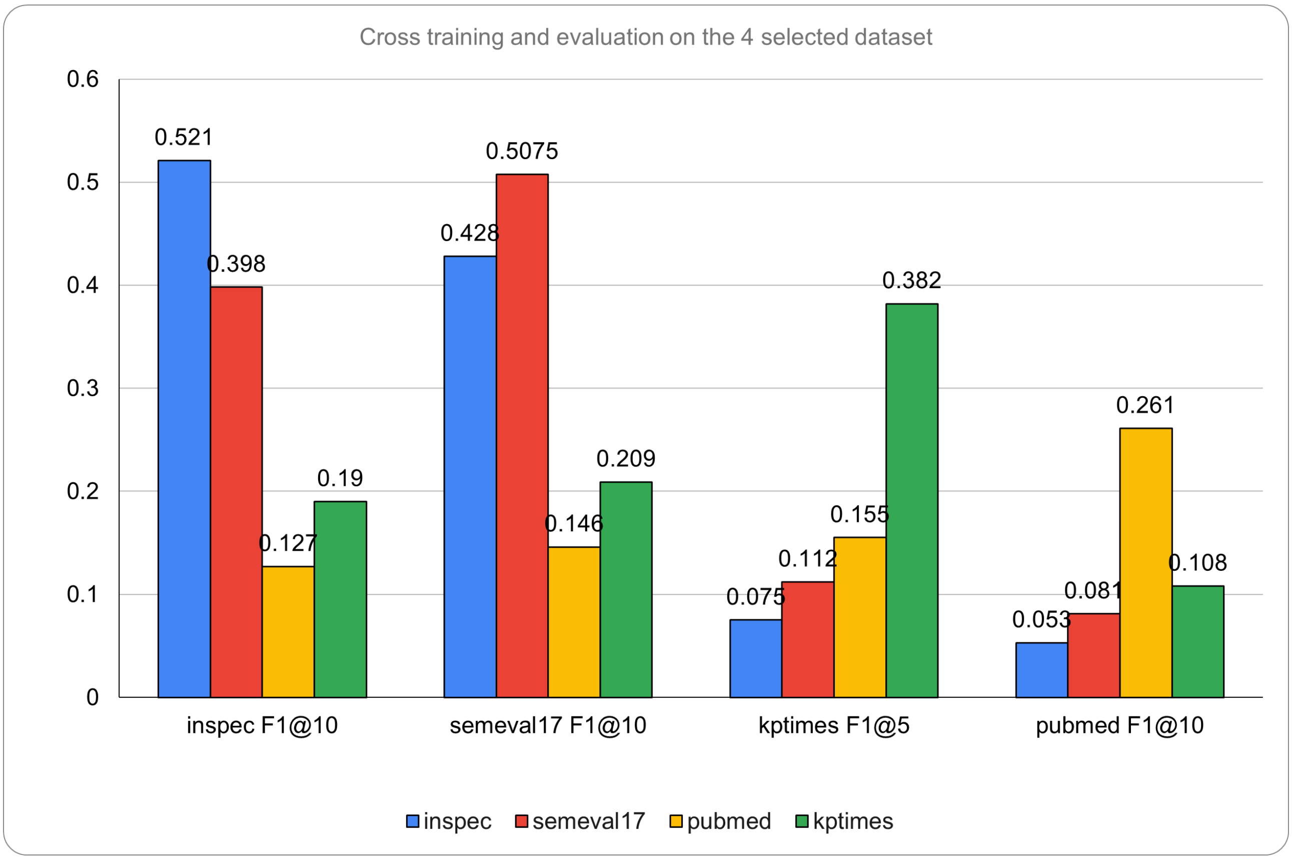

Figure 9: Result of models training on one dataset and evaluating on different datasets.

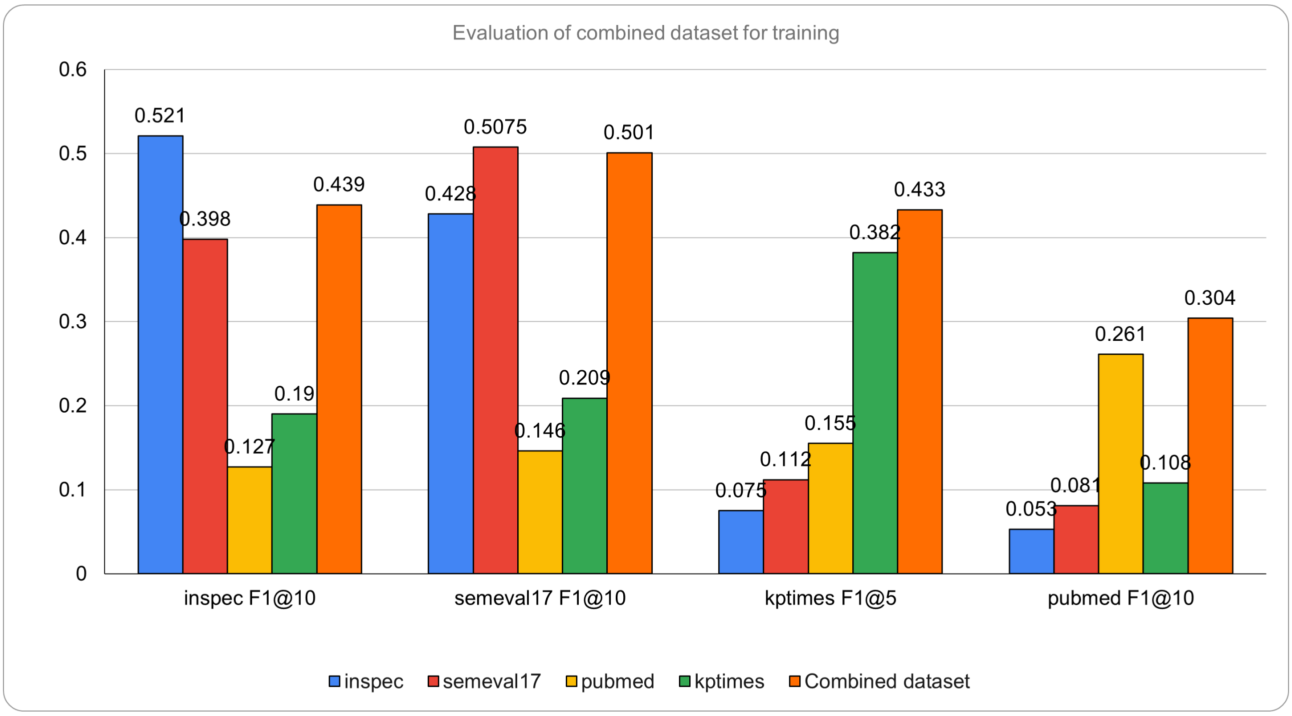

Figure 10: Result of model training by combined dataset

This phenomenon also proves the generalization of the model. The model can learn not only one distribution but several distributions to conclude the logic of keyword extraction. From now on, the dataset for training is the combination of 4 datasets: Inspec, Semeval17, Pubmed, and, Kptimes.

2.4.2 Data splitting from long document

Motivation. The disadvantage of the pre-trained deep language model (BERT) is the limitation in the length of the sequence. For now, the model can only process on the sequence with a length fewer than or equal to 512 tokens. However, datasets Pubmed and Kptimes contain a lot of long documents with lengths larger than 512 tokens. In the last experiments, the model is trained only on the first 512 tokens of the document. Hence, we lost the information on the rest of the document. To use this information for training, we split a long document into several smaller ones with a length of 512 tokens. This action will increase the number of samples dramatically for training. In the data science theory, the more training samples we have, the better model we achieve. This technique is applied to due with long documents in the training set. There are always problems in applying the algorithm to long documents in evaluation.

By the figure 5, there are only two datasets containing long documents: Pubmed and Kptimes. The question is *How to split the long document ?*. We propose two ways of splitting long documents :

- Intuitively, splitting a long document into several consecutive passages without overlapping is the easiest way. We call this splitting as *normal split*.

- , However, the first way of splitting has a disadvantage. In order to learn if a phrase is a keyword, the model needs the context of the phrase. The context of the phrase is the information around the phrase. Hence, if the phrase is located at the beginning or the end of the passage, the model can not obtain sufficient information for training. By this assumption, we propose the second way of splitting: split the long document so that the keywords' positions for each passage are always in the center of the passages. We name this method as *splitting by position*.

These two strategies are described in the figure 11.

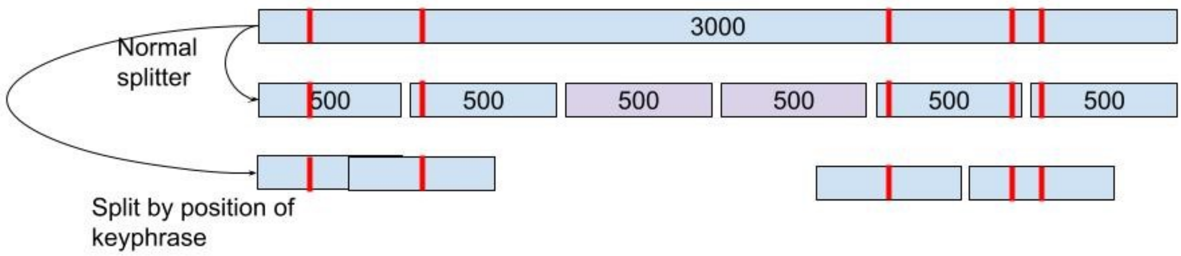

Figure 11: Description of *normal splitting* and *splitting by position*. This is an example of applying two method to split a long document with 3000 tokens and size of passage is 500 tokens. The red mark represent the position of keywords

These two methods dramatically increase the size of the training set. The figure 12 presents the training set size without splitting long documents, normal splitting, and splitting by position. As the normal splitting makes no overlapping between the passages, its size of training is more minor than once of splitting by position.

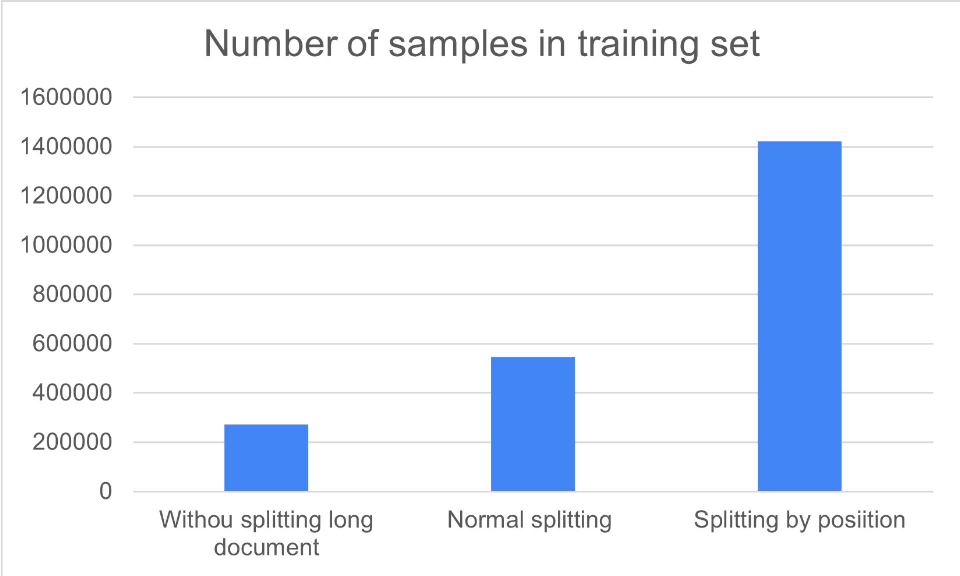

Figure 12: Size of training set by number of samples.

Result. We use these three training sets for training and evaluate on the test set of each component dataset (there are only short documents to evaluate). We obtain the result in the figure 13. There is a considerable improvement between the training set without splitting and the training set with the normal split. Hence, with more training samples, the model is improved. Conversely, although splitting by position makes three times more examples than normal splitting, both results are barely different. It proves that splitting by position can not well improve the model. In conclusion, from this experiment, we use the training set applying normal splitting to train the model.

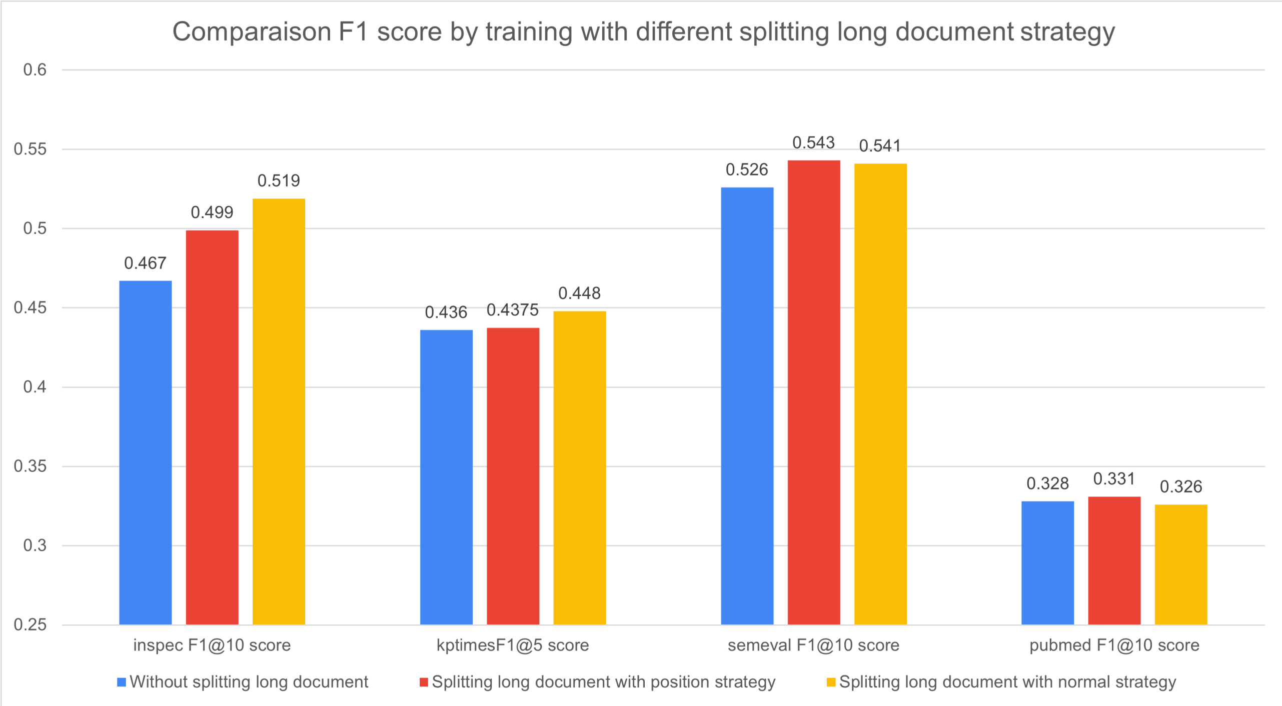

Figure 13: Comparison between performance of training on different strategies of splitting long document.

2.4.3 Casing to Uncasing

As the data pipeline explains, the text is tokenized before being fed into the model. Tokenization[17] is the process of tokenizing or splitting a string or text into a list of tokens. One can think of a token as parts, like a word is a token in a sentence, and a sentence is a token in a paragraph. Bert-joint KPE used Word Piece[18] for tokenization. In Word Piece, there is an option of casing or uncasing. In casing, the tokenization will consider the same word in lowercase and uppercase as two different words. Conversely, in uncasing mode, the text will be lowercased before tokenization. For example, in casing mode, "*We*" and "*we*" are considered as two different words while they are both seen as "*we*" in uncasing mode.

Motivation. In the last experiments, the text is preprocessed with tokenization in casing mode. Hence, the model will see the identical phrase in lowercase and uppercase as different words. It increases considerablyánumber of phrases for calculating by the model. We have the idea to reduce this number by changing the tokenization from casing mode to uncasing mode.

Result. In this experiment, we not only change the tokenization from casing to uncasing but also the pre-trained Bert model from Bert-base-cased to Bert-base-uncased. Bert-base-uncased is the Bert model that is trained by text tokenized in uncasing mode. The figure 14 show the improvement in performance for most of datasets. we get a considerable improvement in the F1 score of Kptimes (0.03), which is the biggest dataset.

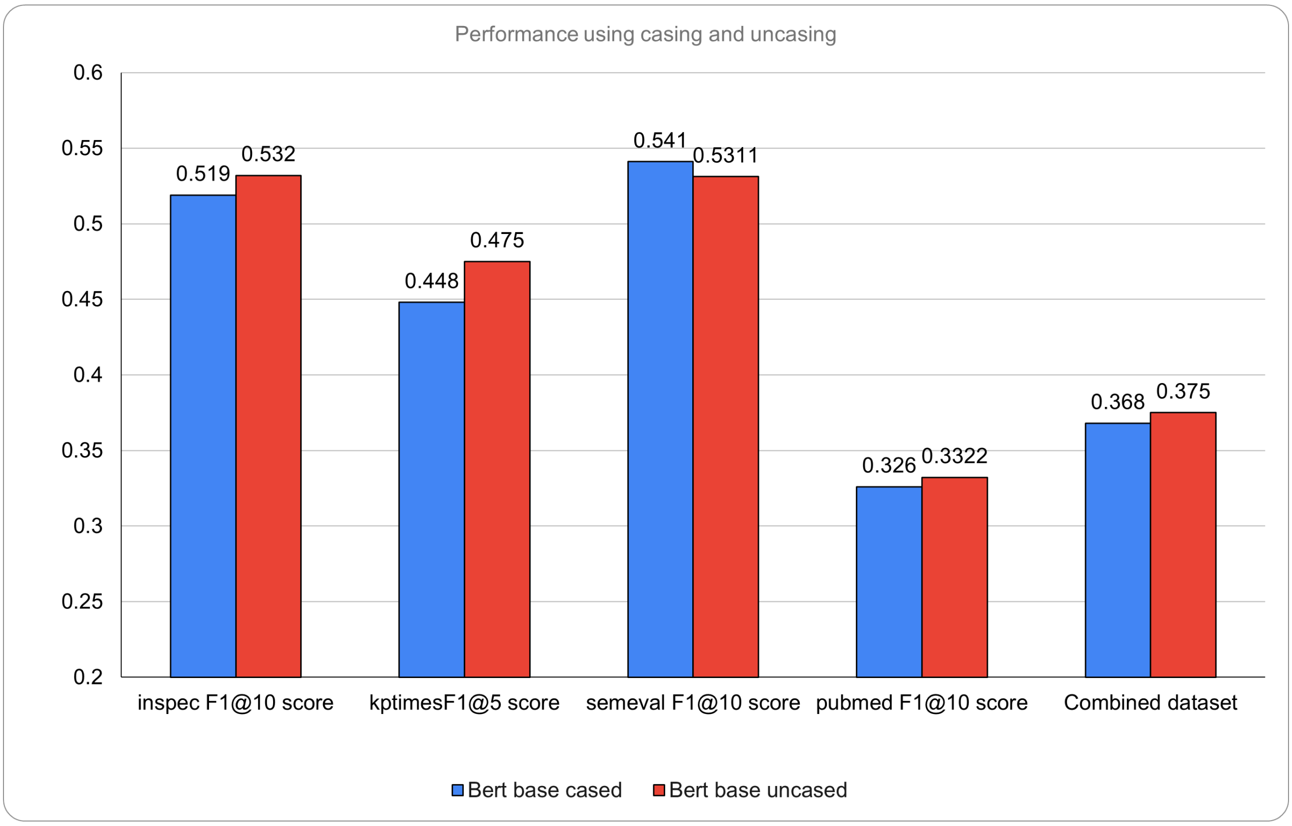

Figure 14: Comparison between casing and uncasing.

2.4.4 Applying variants of Bert

Motivation. Sinequa's product will use the model, it is crucial to minimize inference time which is the time to calculate on a document. By observing each component of the model's architecture, we see that the deep language model takes up most of the parameters in the model, with 110 million on 124 million (88.7%). Hence, the idea is to replace the deep language model with another variant of the Bert model with a smaller size.

As described in the model architecture section, Bert is used in the first part of the model. BERT is designed to help computers understand the meaning of ambiguous language in the text by using surrounding text to establish context. BERT's model architecture is based on Transformers[19]. It uses multilayer bidirectional transformer encoders for language representations. Based on the depth of the model architecture, there is various type of Bert model. The current Bert model is $Bert_{base}$ with 12 layers of transformers block with a hidden size of 768 and number of self-attention heads as 12 and has around 110M trainable parameters. Changing the number of transformers layers and hidden size will change the size of the Bert model. The figure 15 indicates the hyper-parameters of different Bert model. we make the experiments by replacing $Bert_{base}$ by $Bert_{medium}$, $Bert_{small}$ and $Bert_{tiny}$[20].

	H=128	H=256	H=512	H=768
L=2	2/128 (BERT-tiny)	2/256	2/512	2/768
L=4	4/128	4/256(BERT-mini)	4/512 (BERT-small)	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	8/512 (BERT-medium)	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	12/768 (BERT-base)

Figure 15: Variants of Bert with different transformer's layers and hidden size.

Result. To evaluate the result, we calculate the F1 score with 3,5 and 10 candidates on a combined evaluation dataset. The result is indicated in the figure 16. As expected, there are losses in the accuracy of the model. It is reasonable. Because there are fewer parameters in the deep language model, it is worse in generalization. However, these experiments aim to evaluate if the model can reduce the time to calculate on a document. Hence, we measure the inference time on one document. The measured time is calculated based on both CPU and GPU[14]. The result is indicated in the figure 17. There is a significant reduction in inference time. However, the loss in the F1 score is considerable. Nevertheless, the experiment proves that a lighter deep language model can reduce inference time.

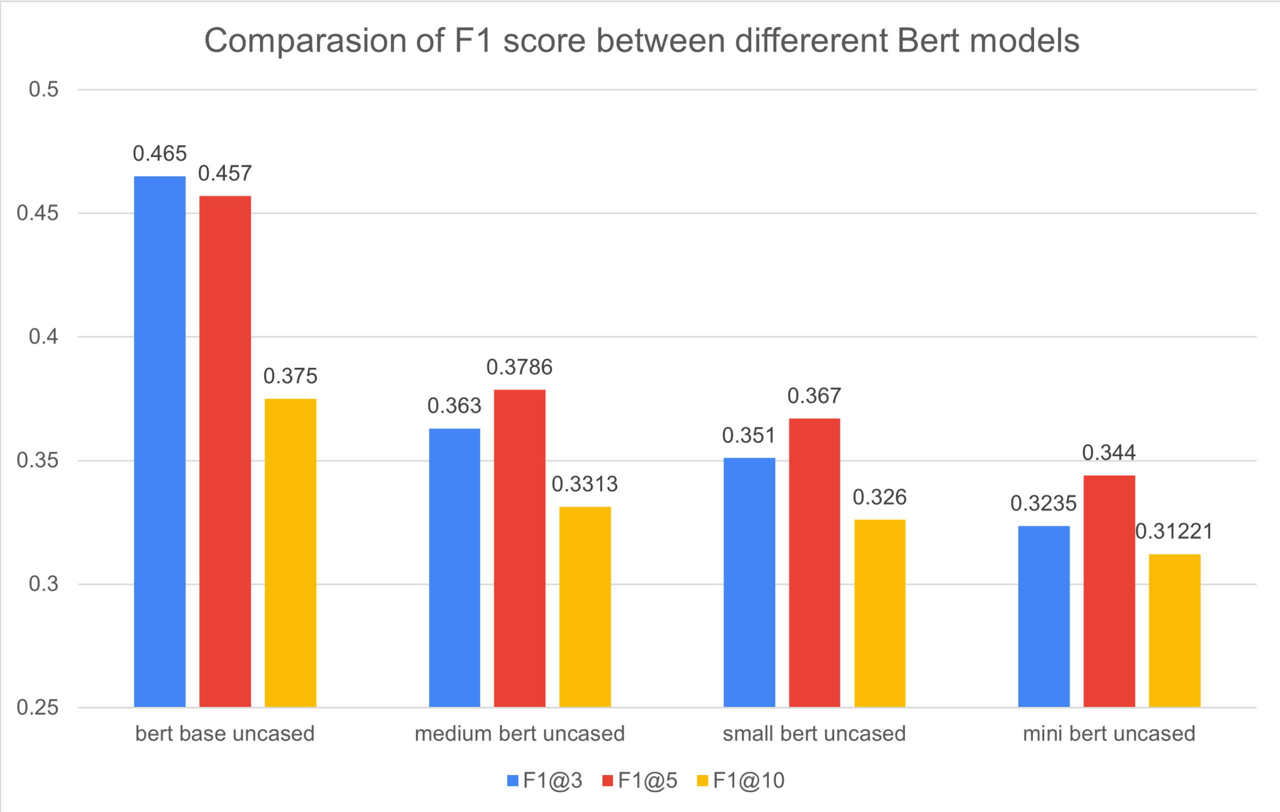

Figure 16: Comparison of F1 score with 3, 5, and 10 candidates between $Bert_{base}$, $Bert_{medium}$, $Bert_{small}$, $Bert_{tiny}$.

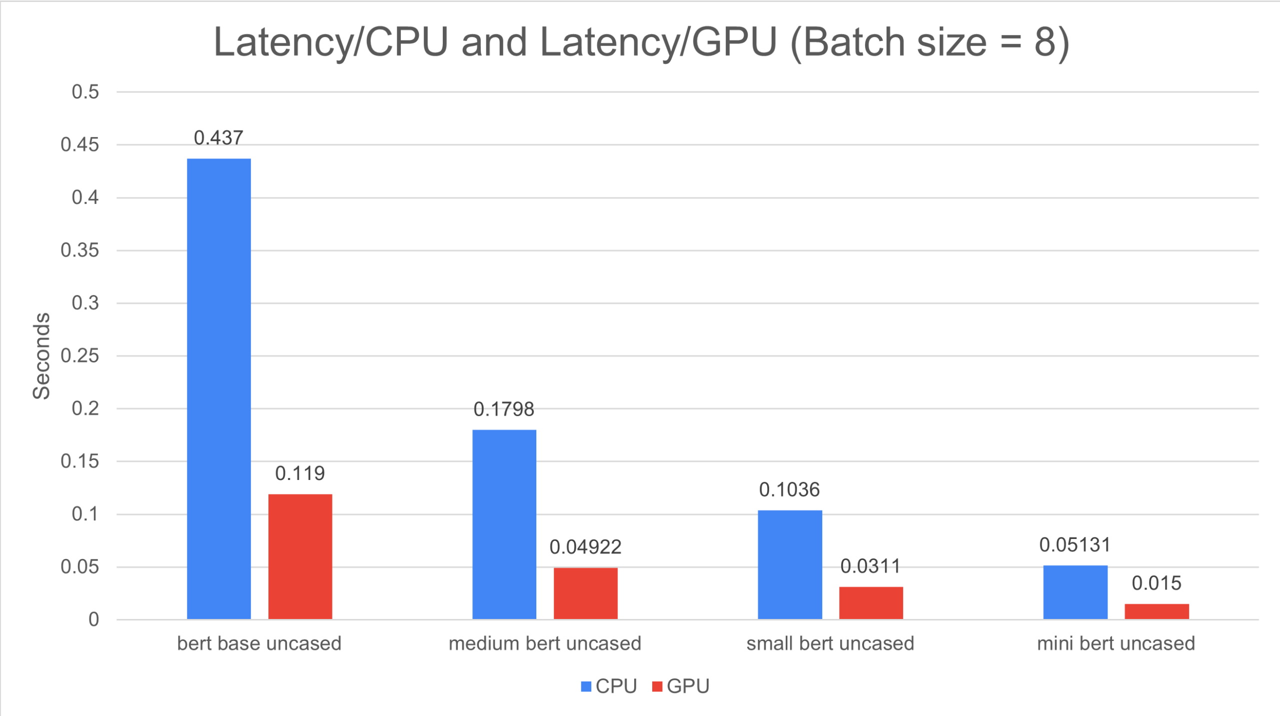

Figure 17: Comparison of inference time on 1 document between $Bert_{base}$, $Bert_{medium}$, $Bert_{small}$, $Bert_{tiny}$.

The above conclusion leads us to experiment with DistilBERT[21]. DistilBERT is a small, fast, cheap, and light Transformer model trained by distilling BERT base. It has 40% fewer parameters than Bert-base-uncased and runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE[28] language understanding benchmark. Hence, with the same idea, we replace $Bert_{base}$ by DistilBERT. The figure 18 and 19 indicate the same metric as above experiments. We can see that, the inference time is reduced by about a half (from 0.437 to 0.276 on CPU, and 0.119 to 0.0682 on GPU) and there is not too much loss in the F1 score. This proves that DistilBERT is a good replacement for $Bert_{base}$.

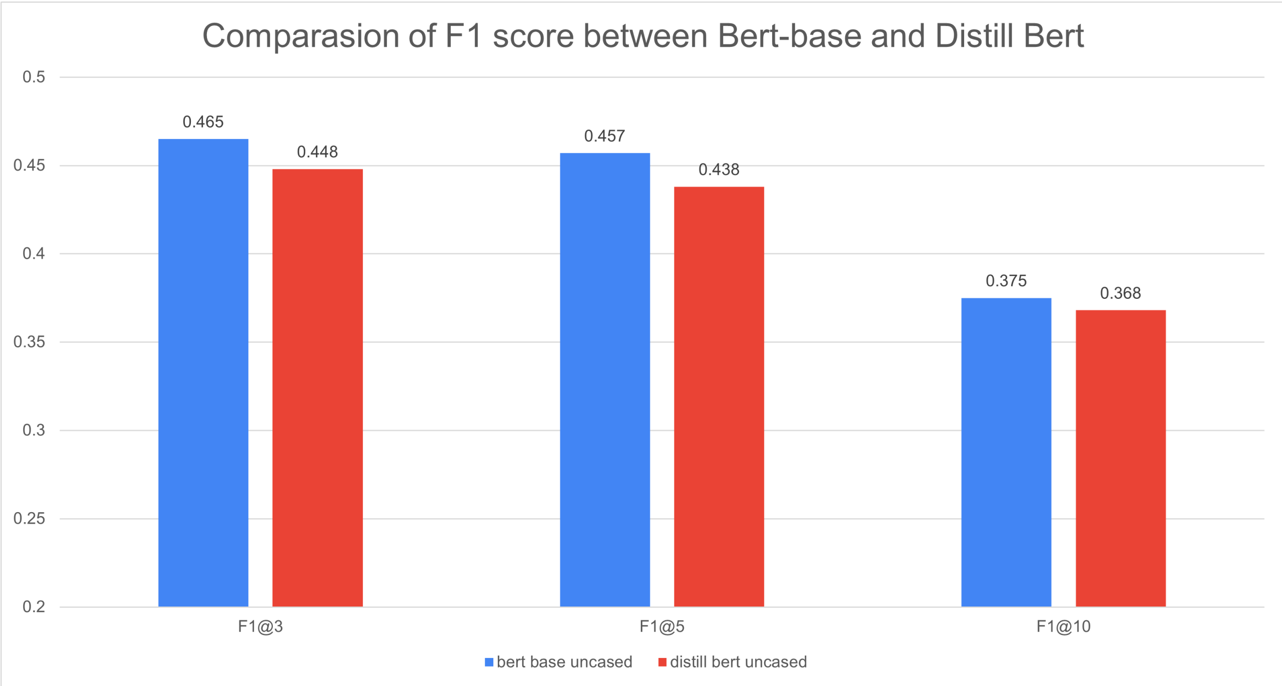

Figure 18: Comparison of F1 score with 3, 5, and 10 candidates between $Bert_{base}$ and DistilBERT.

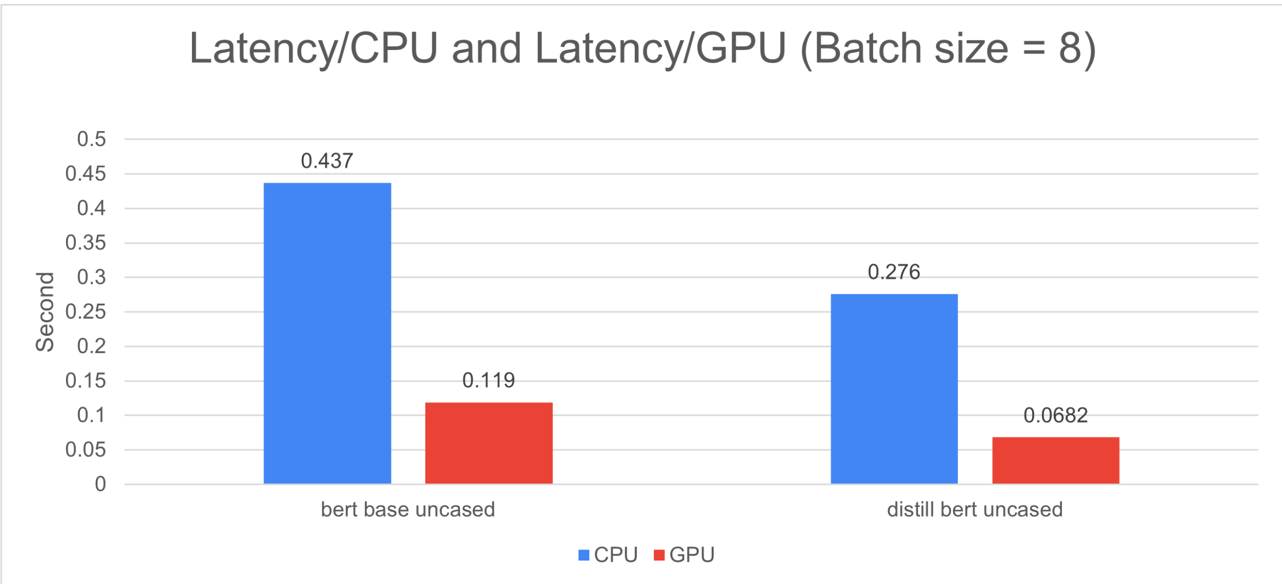

Figure 19: Comparison of inference time on 1 document between $Bert_{base}$ and DistilBERT.

2.5 Research on applying model on long document

Bert base model has the limit for the maximum length of the sequence. As memory complexity would increase quadratically by the increase of sequence's length, the limit length for document fed into the model is 512. Although this limit is suitable for most researched datasets, it does not apply to real-life documents. To resolve this problem, we have two principal directions:

- Using the variants of Bert that can apply for a long document like Longformer[22], Reformer[23], Big Bird[24]...
- Using a slicing window of the model through the text and applying post-processing to conclude the result from the result of each window.

The first direction has some limitations. At first, although some large Bert models can perform on a long document, there is always a maximum for sequence length, for example, 512 or 1024 tokens. However, the real-life problem does not have this limit. Hence, this direction does not resolve the problem. Secondly, a more extensive deep language model will dramatically increase the time for training, evaluating, and inference models. Hence, this is not a good trade-off to pursue.

The second direction also poses advantages and disadvantages. At first, we can apply our model directly without re-training a new one that works only for a long document. Secondly, in most cases, if the passage is long enough, its keywords are also the document's keywords. Based on these hypotheses, if we rightly extract the keywords for each passage, we can predict good keywords for the document. Conversely, the principal drawback is that the deep language model does not cover all documents. Hence, embedding each token does not consider the contextual information on the entire document but only a passage from the document. Hence the solution requires a good strategy to split the long document into several passages. Another problem with a long document is aggregating each passage's results into one result that raises the best performance. After researching and considering both methods, we choose the second direction to continue our research on a long document.

The following steps describe the process for a long document:

- Given the long documents (length 512 tokens), we split the document into several passages with a length of 512. However, two consecutive passages overlap. The length of overlap is a parameter to fine-tune.
- Apply Bert-KPE on each passage to extract a list of keywords and the correspondent score. We propose the number candidate of each passage as a parameter to fine-tune.
- Construct a map of keywords and the scores correspondent. The keywords are the prediction on each passage, and the score is the sum of the score for repeated keywords.

- Sort the keywords by their scores and get the top candidates as the result of the document. Maximum candidates are considered as a parameter to fine-tune.

The process has three parameters to fine-tune: length of overlap, number of candidates for each passage, and maximum candidates for each document. After fine-tuning these hyper-parameters, we conclude the following set of hyper-parameter :

- Length of overlap = 200 words.
- Number of the candidates for each passage = 3.
- Maximum candidates for each the whole document = 10.

We compare the performance with the native strategy. The native strategy makes the model calculate only on 512 first tokens. This is the default strategy of paper for a long document. Figure 20 shows the improvement from native strategy to research strategy. We evaluate Pubmed and Kptimes (two datasets contain long documents). We obtain a significant improvement in both datasets.

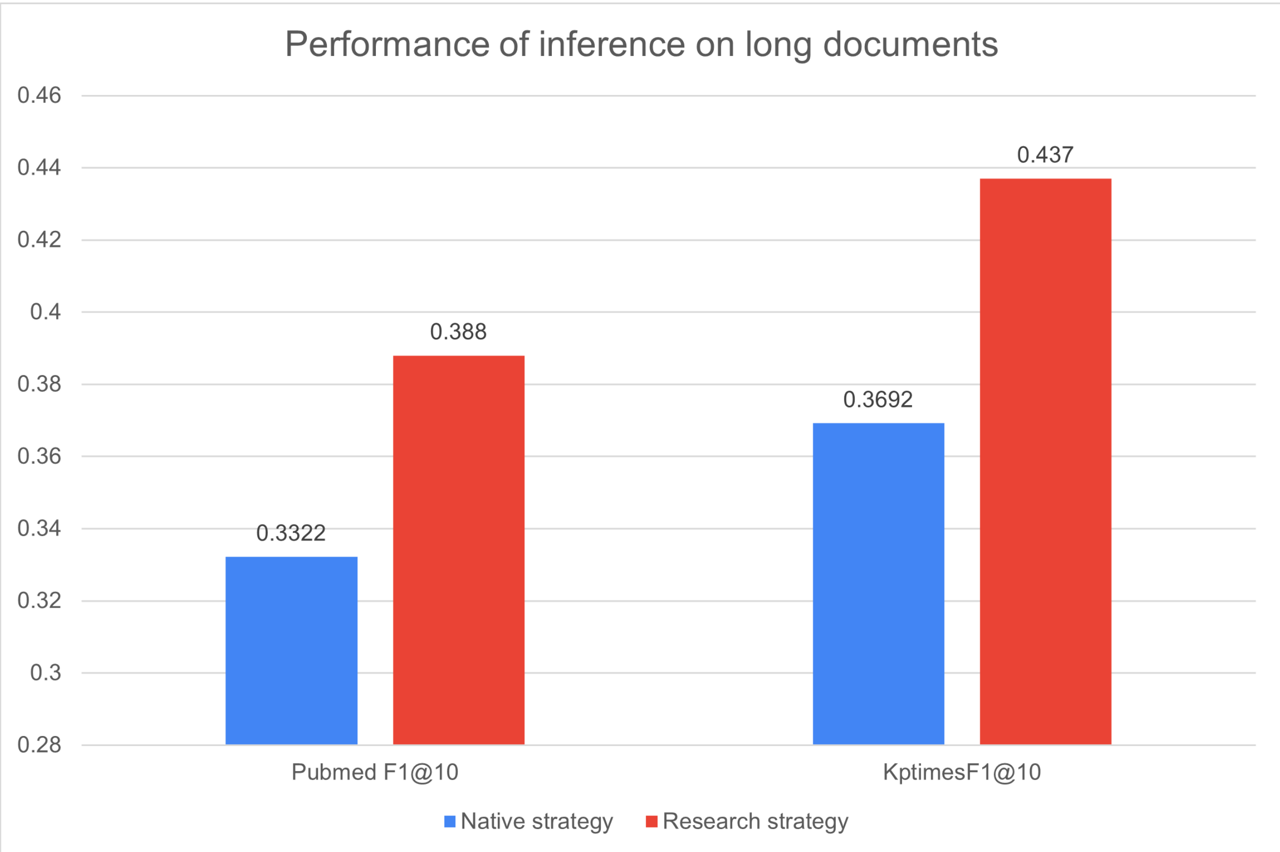

Figure 20: Comparison of inference on long documents between native strategy and researched strategy.

2.6 Conclusion

In conclusion, in the first part of my internship, we research to resolve the problem of Keyword extraction. The researched paper is Bert Joint KPE. This is a deep learning algorithm. The paper proposes an architecture using the Bert-base model to understand the context of the document and a convolutional network to calculate the representation of each phrase. The paper also uses two loss functions to train the model: *loss of chunking* for classification if a phrase is keywords, and *loss of ranking* to recognize the difference between a keyword and a usual phrase.

Various datasets for keyword extraction are explored in this phase. We conclude with four datasets that are available for commercial use. Analysis of these datasets shows some properties of keywords. It also indicates that there are differences in annotating keywords between different sources. We construct a data pipeline to transform data from these datasets to a standard format of input for the model.

To research the model deeply, we re-implement the algorithm with a new framework. This framework is used regularly at Sinequa. To evaluate our implementation, we train and evaluate the model on the same dataset, with the same hyperparameters as the paper's and achieve relatively similar performance. Hence, we conclude that the implementation can be used to research the algorithm.

Training and evaluating the model on the combined dataset is our first achievement. Concatenating all four datasets gave us a generalized training set and evaluation set. Evaluating the combined dataset gives a better metric for evaluating the model's performance. Training on a combined dataset considerably improves the performance of the model.

Bert base model is not trainable with a long document (> 512 tokens). Hence we split a long document into several passages. This idea increases the number of training samples. Hence it improves the model performances from 0.322 to 0.368. Our next idea is to replace the tokenization setting from casing to uncasing. It reduces the number of phrases seen by the model. It improves the model's performance from 0.368 to 0.375 on combined dataset.

As the model will be used in production, it should not only be exact but also fast. To improve the calculation speed, we replace the deep language model BERT with a lighter version of DistilBERT. This reduces the inference time on one document from $0.119s$ to $0.0682s$ on one document, and the loss in F1 score is acceptable.

The algorithm works only on documents with length < 512 tokens. Hence it is not suitable for real situations. We expand the algorithm and propose a post-processing strategy to calculate and conclude the keywords for long documents. The strategy achieves 0.437 of F1 score at ten candidates in the long document dataset Kptimes and 0.388 on Pubmed with the same metric.

The result of my first part includes a Python script and a strategy to train and evaluate the model, a model file that will be used in production, and a strategy to apply the algorithm in a long document.

3 Implementation in Sinequa's production

3.1 Introduction

Sinequa Enterprise Search Platform is the main functionality of Sinequa's production. An enterprise search platform must serve most people in an organization with information and insights extracted from its vast amounts of data. The Sinequa Enterprise Search Platform makes it easy to connect to all of that data – enterprise applications (CMS, ERP, CRM, PLM, etc.), files, documents, and email in different languages. It provides a simple unique interface to deliver information and insights to everyone without requiring particular expertise. The figure 21 represents an example of Sinequa's interface.

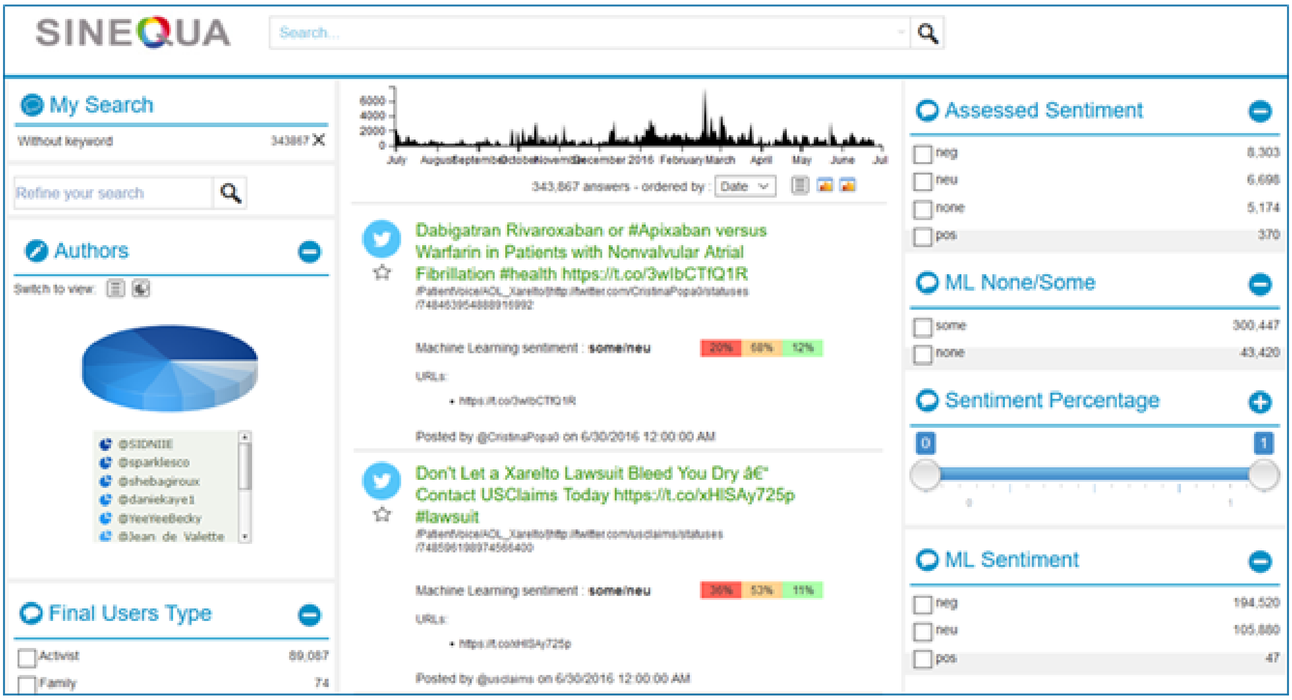

Figure 21: Sinequa software interface.

So far, before the use of machine learning at Sinequa, most information extracted from the text came from the work of a linguistic team that mainly used regular expressions to extract, for example, entities. In the next step of development, Sinequa's product must be able to extract as much information as possible in one document automatically. This information includes keywords, topic, etc... And the project is named *Context augmentation*. With this feature, the application can better understand the document. Hence, it is expected to improve the search result and user experience. **Integration of keywords extraction** is the first component of the whole project.

3.2 Implementation detail

The main purpose of integration is to apply the model from the research part. There are three main steps to conclude the keywords from the raw text :

- - Preprocess the text to obtain the format of training text in python.
- - Load the model from the file, and calculate the model's output from the preprocessed text.
- - Post process model's output to conclude the keywords of the text.

These steps are described in the section **Pre-processing and Post-processing** and **Model serving**. Section **model manager and Keyword Extraction API** describes the management of multiple models simultaneously and the API using this manager. Then, We implement the feature into two main functionalities of Sinequa software described in the section **Keyword Extraction in Serving Routes and Sinequa Command**. The figure 22 represents the whole implementation.

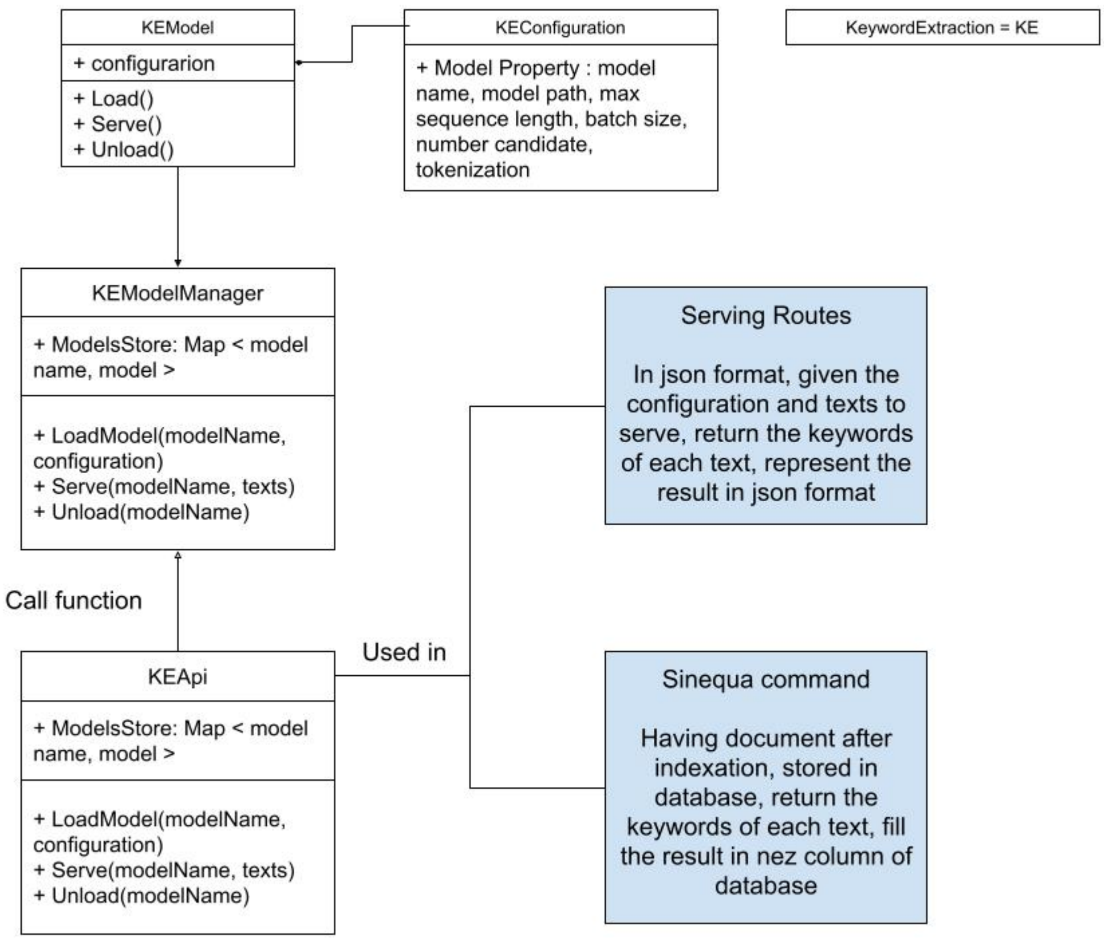

Figure 22: Implementation's architecture.

3.2.1 Pre-processing and Post-processing

Preprocessing is the step to transform from raw text to the format the model is trained on. As Sinequa software can work on enormous types of documents, there are a lot of ambiguous characters. Hence, preprocessing limits the type of character that the model can read.

The preprocessing includes :

- Lowercase all the character.
- Change *new line* character to space. Based on the operating system, this character can be $\backslash n$, $\backslash t$, $\backslash w$...
- Replace consecutive *space* characters to one *space* character
- Filter the character which are out of this list : A - Z, a - z, 0 - 9, !,@,#,$,%,&.*,(),[],,+,-,/.

The image 23 represent an example of preprocessing for a raw text.

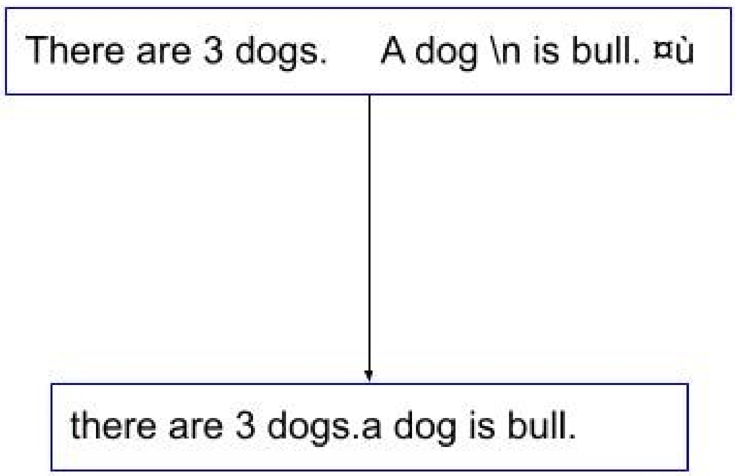

Figure 23: Preprocessing example.

Post-processing is the step to conclude the keywords from the output of the model. In fact, model's output is just a list of score. Each score represents the probability that the corresponding phrase is a keyword. The post-processing includes :

- Get top k phrases with highest scores.
- Filter the *duplicated phrases by meaning.* Duplicated phrases by meaning are the phrases which are different by spelling but same by meaning. For examples : "dogs" and "dog". The main technique for this step is stemmization[26]. Stemmization transforms a word to its original form.

The image 24 represents an output of an model and the post-processing to conclude 3 keywords.

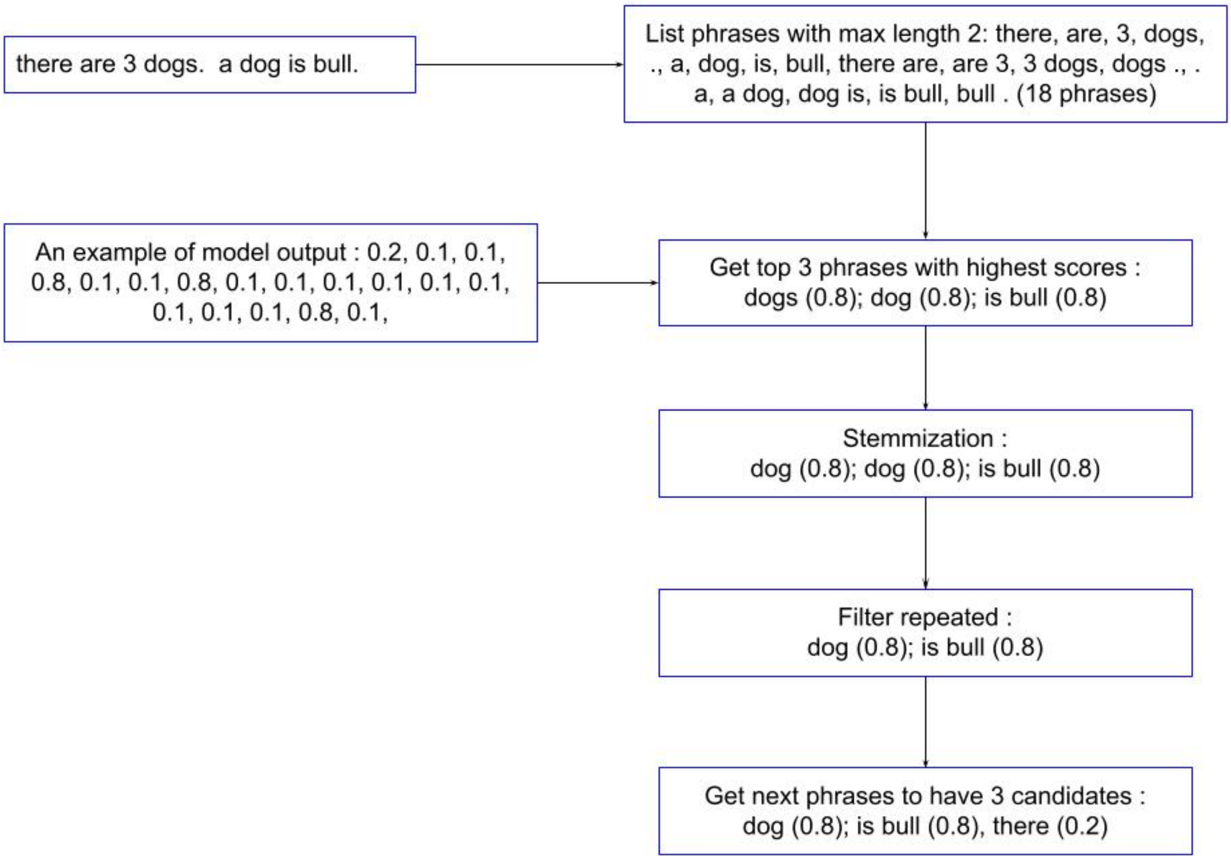

Figure 24: An small example of model's output and post-processing to conclude 3 keywords from documents.

3.2.2 Model serving and model manager

Model serving is the principal component of the integration. The main functionality is to calculate from preprocessed text to the output. These are five steps of model serving :

- Batchify a list of text.
- Build input feature of the model.
- Load model from the file using a configuration.
- Calculate the output.
- Unload model

Batchify. To speed up the computation, we benefice from the ability of the deep learning model. We apply the model to a batch of documents. Hence the application can calculate on several documents at the same time. This technique is called batchify the input. The figure 25 represents the batchify with batch size = 4. Hence, we can calculate four documents at the same time.

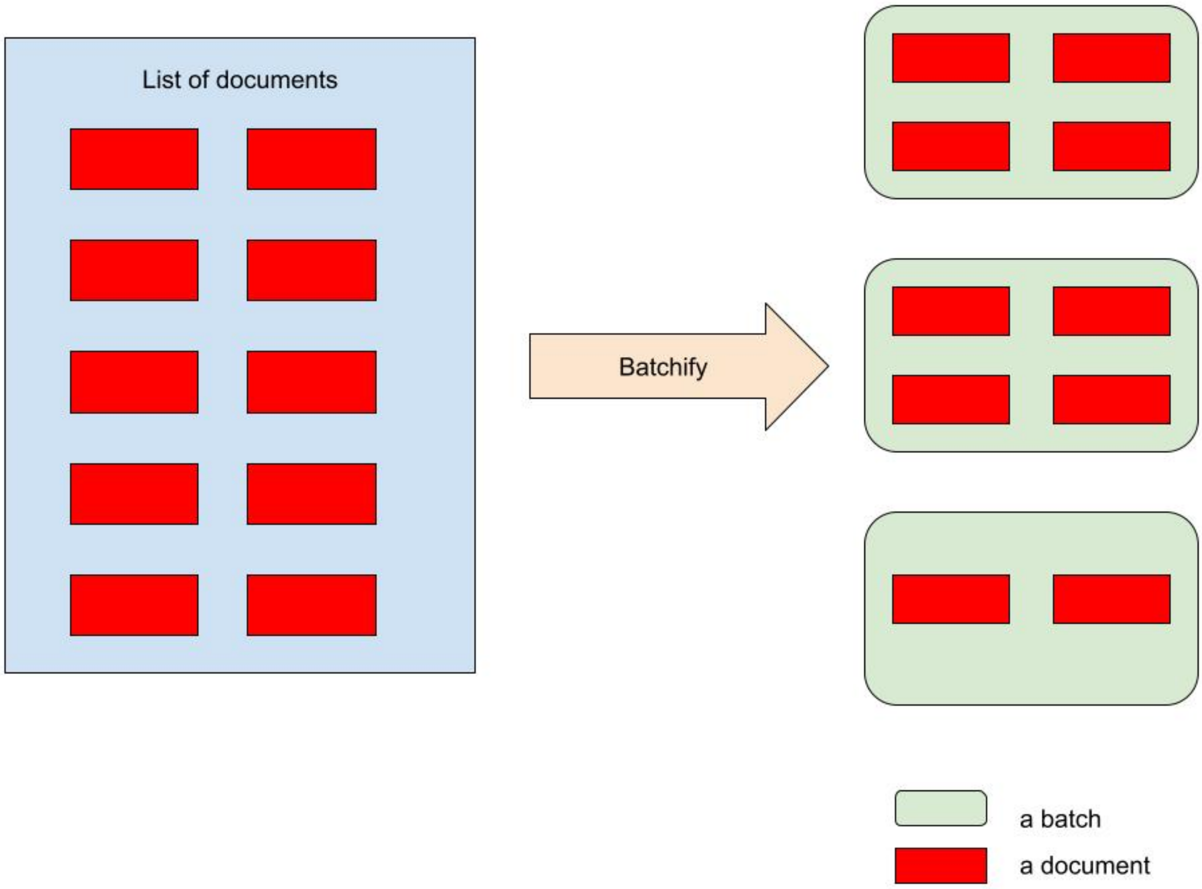

Figure 25: Batchify a list of 10 documents with batch size = 4

Build input feature. input of the model is not the text but a very complex matrix. The matrix includes the information of token ids from text and the position of every phrase of the document.

Load model using a configuration. The model needs a configuration. Configuration is a list of parameters. There are two types of configuration: hard-coded config and user config. Hard-coded config is the constant parameters of the model like path to the model file, max length of the sequence, max length of phrases, ... User config is the parameter that can be changed by users like the number of candidates to extract, computational unit (CPU or GPU). After the configuration, the model can be quickly loaded by a class well developed by the ML team.

Calculate the output. We can calculate the model's output using the input feature and the model in the last two steps. Then, we can conclude the keywords of the model using Post-processing.

Unload model. After having the keywords, unloading the model is important to release the memory.

3.2.3 Model Manager and Keyword Extraction API

Sinequa application must be able to use multiple models in parallel. Although we use only one model, for now, the project keyword extraction will expand and need this feature in the future. **Model manager** is a map between a name and a model. Like a model, a model manager has three main functions: Load model, Serving model, and Unload model. Each one also receives a name so the manager can detect which model is using. The figure 26 represents the idea of model manager.

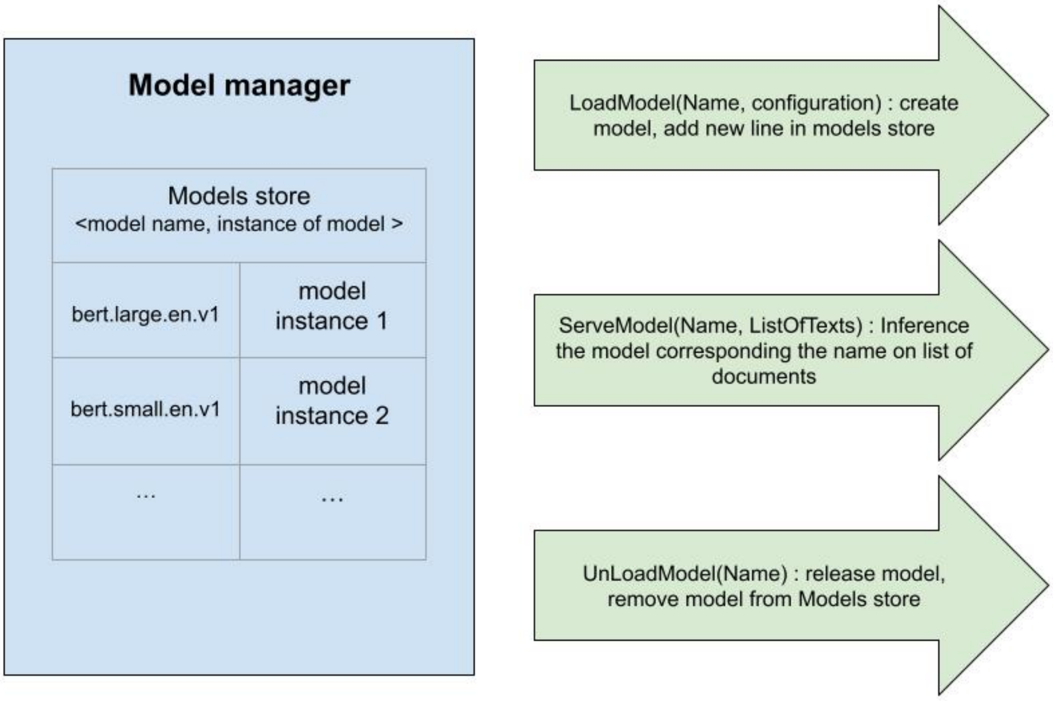

Figure 26: Model manager and its 3 main functions

An API is a set of programming codes that enables data transmission between one software product and another. In our case, the API is the interface of the keyword extraction feature. Hence, other components of the software can use this feature.

3.2.4 Keyword Extraction in Serving Routes and Sinequa Command

In this section, I describe 2 functionalities of Sinequa application using keyword extraction : *Serving Routes* and *Sinequa Command*. These functionalities are implemented based on the API in the above section.

Serving routes is the method to use keyword extraction manually. Postman[27] is the application that helps us test the serving routes. We define a configuration of the model and a list of texts in JSON format. Then we send this information as a message to the address of the Keyword extraction API defined by Sinequa software. Postman returns a list of keywords and corresponding scores for each keyword. The figures 27 and 28 are an example of this functionality.

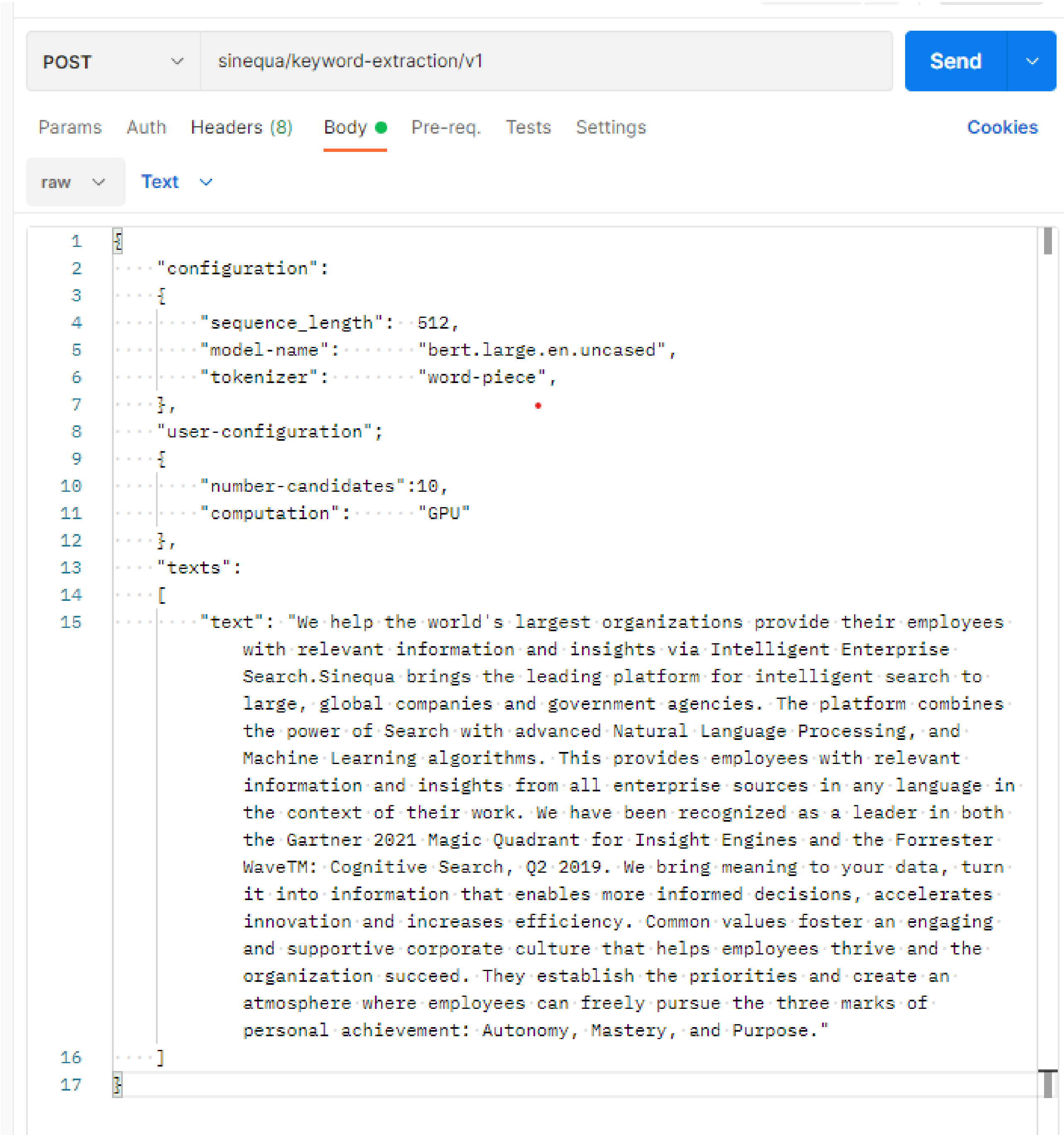

Figure 27: An example of Postman request for keyword extraction. The text is an introduction of Sinequa's official website[1]

```
Response:
{
    "keyword_1" : intelligent enterprise search
    "score": 3.2452
    "keyword_2 : sinequa
    "score": 3.2421
    "keyword_3 : data chaos
    "score": 3.1733
    "keyword_4 : natural language processing
    "score": 3.0155
    "keyword_5 : innovation
    "score": 2.9745
    "keyword_6 : relevant information
    "score": 2.7853
    "keyword_7 : leader
    "score": 2.4428
    "keyword_8 : efficiency
    "score": 2.4191
    "keyword_9 : cognitive search
    "score": 2.0331
    "keyword_10 : organization
    "score": 1.9956
}
```


Figure 28: An example of Postman response for request of figure 27

Sinequa Command is the method applied inside the application. In brief, to perform a search platform on an enormous amount of documents, the software indexes each one, and save all of them into an SQL table. Each document is saved in a row on the table. These documents are called *Indexed documents*. Sinequa command performs keyword extraction on indexed documents and saves the result in the corresponding row of the document. The figure indicates the step where

Sinequa command *keyword extraction* is used

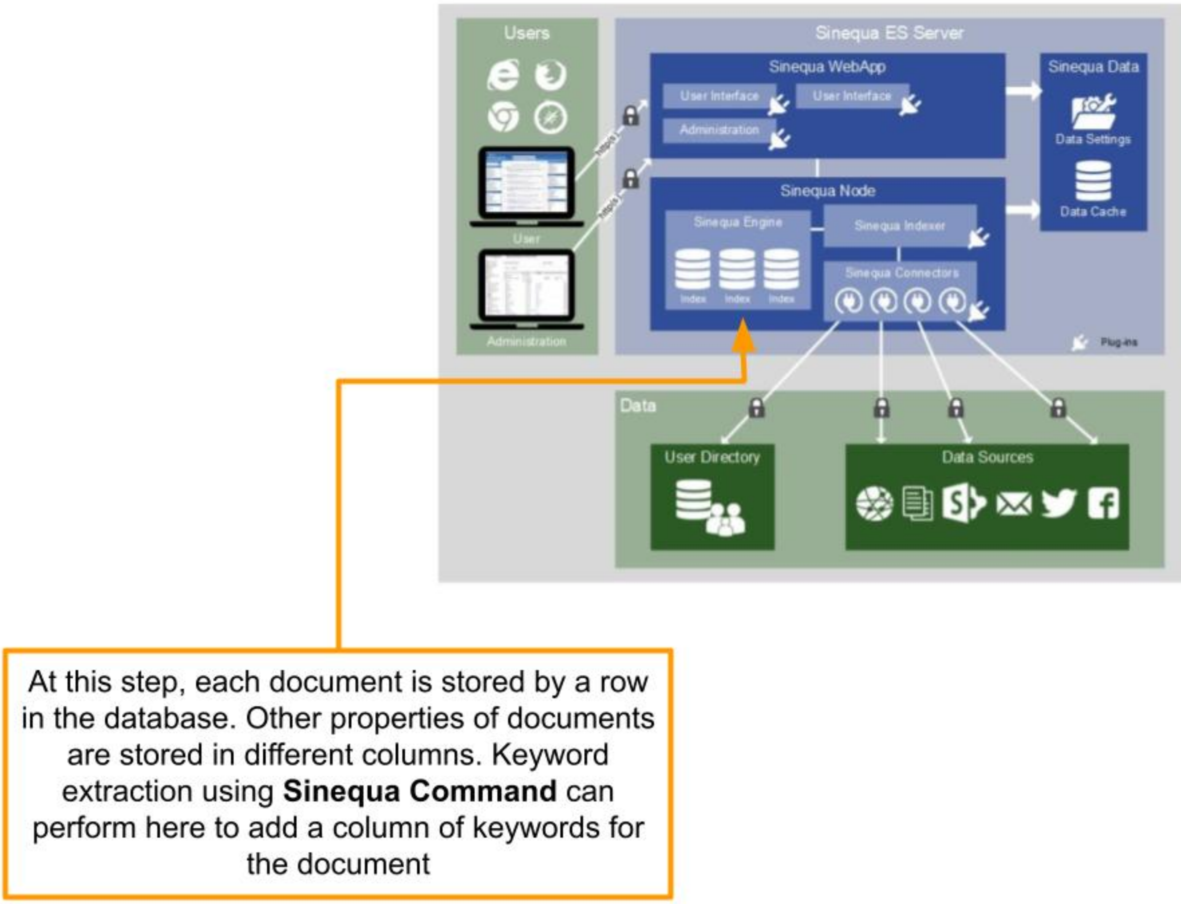

Figure 29: Sinequa command of keyword extraction in the context of Sinequa scheme

These are the steps to perform Sinequa Command :

- Define a configuration for keywords extraction.
- Define a command. This definition include :
  - Choose the configuration to apply.
  - Choose the indexed documents to perform keywords extraction.
  - Define a column to store the result which is list of keywords for each text.
- Using SQL command to verify the result of command.

3.3 Conclusion

In conclusion, the keyword extraction feature is implemented into the Sinequa software. This is the second part of my internship. I design a structure by the method of Oriented Object Programming. There are 4 main components: pre-processing, model configuration, model serving, and post-processing.

- Pre-processing is the transformation from raw text to the input format of the model.
- Model configuration is a set of parameters to configure the model.
- Model serving includes: batchify the input, build input feature, load model from a file, calculate the model output and unload the model
- Post-processing is the process of concluding the keywords from the model's output.

Then, I expand my integration to models manager. This design helps us manage multiple models at the same time. The keyword extraction API is implemented based on the model manager. The API is the interface of the keyword extraction feature. This feature is integrated into two main functionalities of Sinequa software: Serving Routes and Sinequa Command. In conclusion, after my integration, users can use the feature of keyword extraction on Sinequa production.

4 Conclusion

During my internship at Sinequa, subjects related to *Keyword extraction using deep learning* were carried out. The research on resolving keyword extraction by the algorithm BERT-jointKPE has significantly progressed. The algorithm was deeply researched and optimized. The training data was adapted to the context of business. The model was improved by: concatenating various data sources, splitting long documents into passages, and using tokenizer uncasing instead of the casing. The model was also optimized in inference time by replacing BERT with DistilBERT. I also proposed a strategy to apply the algorithm to long documents to overcome the limitation of the sequence length of the Bert-base model.

However, there are still jobs that have not been processed yet. The research can be expanded into another language than English. The model's training process can be sped up by using Google Colab TPU. A recent new dataset LDKP10K is a free commercial dataset for keyword extraction. This dataset could be an excellent distribution to generalize the model.

The subject of integration keyword extraction feature in Sinequa software has been completed. The feature includes four components: pre-processing, model configuration, model serving, and post-processing. Model serving is the most complicated component. It includes five steps: batchify, build input feature, load model, calculate the model output, and unload model. The feature was expanded into a design that manages multiple models simultaneously. The model manager was used to implement Keyword Extraction API, the feature's interface. Then, this API was used in 2 main functionalities of Sinequa production: Serving Routes and Sinequa Command.

Moreover, this internship gave me much experience. At the technical level, I had the chance to research a complex problem, apply my knowledge in data science, and develop a huge code base product. Furthermore, as a machine learning engineer, I could watch, work, and understand the work. At the non-technical level, I had the opportunity to discover the work of a large technical team, the structure of a technology company, and the methodology for managing a project.

As my internship is very much related to research in deep learning and software development, I better understood the work in this field. It helped me shape my path in the future. The techniques covered in the internship are very up-to-date, and the problems I encountered allowed me to develop my problem-solving skills and learn basic knowledge and techniques and experiences.

Anyway, I had an excellent opportunity to work in the ML team of Sinequa, under the instruction of a Machine Learning Engineer, with exciting people in the field directly related to my future career.

References

[1] *Introduction of Sinequa*: Information on official website of Sinequa.
https://www.sinequa.com/

[2] *TF-IDF*: Term frequency-inverse document frequency
https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[3] *Sinequa product*: Information of Sinequa's main production
https://www.sinequa.com/product-enterprise-search/

[4] *Gartner magic quadrant for insight engines* : Definition of Gartner magic quadrant for insight engines.
https://www.gartner.com/en/documents/3999454

[5] *Sinequa customer*: Official clients of Sinequa.
https://www.sinequa.com/customers/

[6] *Keywords extraction*: Introduction of keywords extraction task.
https://monkeylearn.com/keyword-extraction/

[7] *Keywords generation*: Introduction of keywords generation task, another track for keyword extraction.
https://dl.acm.org/doi/abs/10.1145/3308558.3313570

[8] *BERT*: Pre-training of Deep Bidirectional Transformers for Language Understanding.
https://arxiv.org/abs/1810.04805

[9] *Inspec*: Science dataset for keywords extraction.
https://www.ebsco.com/products/research-databases/inspec

[10] *Semeval17*: ScienceIE - Extracting Keyphrases and Relations from Scientific Publications.
https://aclanthology.org/S17-2091/

[11] *Pubmed*: Keyword extraction and structuralization of medical reports.
https://pubmed.ncbi.nlm.nih.gov/32269770/

[12] *Kptimes*: A Large-Scale Dataset for Keyphrase Generation on News Documents.
https://aclanthology.org/W19-8617.pdf

[13] *Bert-JointKPE*: Capturing Global Informativeness in Open Domain Keyphrase Extraction.
https://arxiv.org/abs/2004.13639

[14] *GPU*: Graphics processing unit.
https://en.wikipedia.org/wiki/Graphics_processing_unit

[15] *TPU*: Tensor Processing Unit.
https://en.wikipedia.org/wiki/Tensor_Processing_Unit

[16] *pytorch-lightning*: Introdduction of Pytorch Lightning
https://towardsdatascience.com/from-pytorch-to-pytorch-lightning-a-gentle-introduction-b371b7caaf09

[17] *tokenization*: Introduction to tokenization.
https://neptune.ai/blog/tokenization-in-nlp

[18] *WordPiece tokenization*: Introduction to WordPiece tokenization.
https://huggingface.co/course/chapter6/6?fw=pt

[19] *transformer*: Attention Is All You Need.
https://arxiv.org/abs/1706.03762?context=cs

[20] *Bert Compact Models*: Well-Read Students Learn Better- On the Importance of Pre-training Compact Models.
https://arxiv.org/abs/1706.03762?context=cs

[21] *DistilBERT*: A distilled version of BERT - smaller, faster, cheaper and lighter.
https://arxiv.org/abs/1910.01108

[22] *Longformer*: The Long-Document Transformer.
https://arxiv.org/pdf/2004.05150.pdf

[23] *Reformer*: The Efficient Transformer.
https://arxiv.org/abs/2001.04451

[24] *Big Bird*: Transformers for Longer Sequences.
https://arxiv.org/abs/2007.14062

[25] *Meta-Learning*: Learning to Learn Fast.
https://lilianweng.github.io/posts/2018-11-30-meta-learning/

[26] *Stemming*: Stemming and Lemmatization.
https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8

[27] *postman*: API platform.
https://www.postman.com/

[28] *GLUE*: The General Language Understanding Evaluation.
https://gluebenchmark.com/

List of Figures

A Annex

A.1 Bert Model

BERT (Bidirectional Encoder Representations from Transformers) is a recent paper published by researchers at Google AI Language. It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks, including Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others.

BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible.

How BERT works

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary. The detailed workings of Transformer are described in a paper by Google.

As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional, though it would be more accurate to say that it's non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

The chart below is a high-level description of the Transformer encoder. The input is a sequence of tokens, which are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors of size H, in which each vector corresponds to an input token with the same index.

A.2 Word-Piece Tokenization

WordPiece is a subword-based tokenization algorithm. It was first outlined in the paper "Japanese and Korean Voice Search (Schuster et al., 2012)". The algorithm gained popularity through the famous state-of-the-art model BERT.

Subword-based tokenization is a solution between word and character-based tokenization. The main idea is to solve the issues faced by word-based tokenization (very large vocabulary size, large number of OOV tokens, and different meaning of very similar words) and character-based tokenization (very long sequences and less meaningful individual tokens).

The subword-based tokenization algorithms do not split the frequently used words into smaller subwords. It rather splits the rare words into smaller meaningful subwords. For example, "boy" is not split but "boys" is split into "boy" and "s". This helps the model learn that the word "boys" is formed using the word "boy" with slightly different meanings but the same root word.

In addition, WordPiece algorithm trains a language model on the base vocabulary, picks the pair which has the highest likelihood, add this pair to the vocabulary, train the language model on the new vocabulary and repeat the steps repeated until the desired vocabulary size or likelihood threshold is reached.

A.3 Distill Bert Model

The DistilBERT model was proposed in the blog post Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT, and the paper DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.

What is distillation? The concept of distillation is quite intuitive: it is the process of training a small student model to mimic a larger teacher model as close as possible. Distillation would be useless if we only run machine-learning models on the cluster we use to fine-tune them, but sadly, it isn't the case. Therefore, distillation comes in whenever we want to port a model onto smaller hardware, such as a limited laptop or a cellphone, because a distilled model runs faster and takes less space.

The necessity of BERT distillation As you might have noticed, BERT-based models are all the rage in NLP, since they were first introduced. And with increasing performances came many, many parameters. Over 110 million for BERT, to be precise, and we aren't even talking about BERT-large. Thus, the need for distillation was apparent, since BERT was so versatile and well-performing. Furthermore, models that came after were basically built the same way, akin to RoBERTa [3], so by learning to properly distill BERT, you could kill two birds with one stone.

Copying the teacher's architecture BERT's mainly based on a succession of attention layers stacked on top of each other. Therefore, it means that the 'hidden knowledge' BERT learns is contained in those layers.

From one BERT to another, the number N of layers varies, but of course the size of the model is proportional to N. It follows that the time taken to train the model and the duration of forward passes also depend on N, along with the memory taken to store the model. The logical conclusion to distill BERT is thus to reduce N.

DistilBERT's approach is to half the number of layers, and to initialize the student's layers from the teacher's. Simple, yet efficient

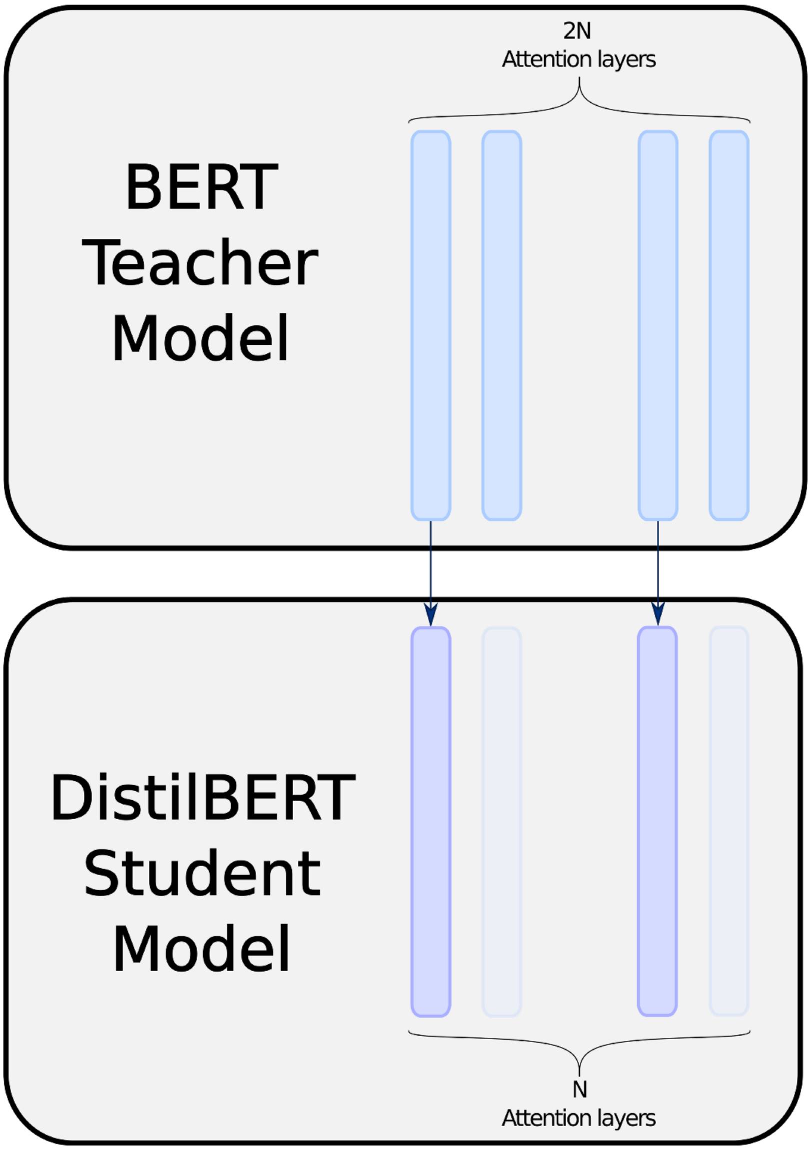

Figure 30: From Bert to DistilBERT

DistilBERT alternates between one copied and one ignored layer which tried copying top or bottom layers in priority.