ASKME : QUESTION ANSWERING USING BERT AND WIKIPEDIA

SOFTWARE PROJECT

Axel Didier
M2 NLP
axel.didier5@etu.univ-lorraine.fr

Pierre Goncalves
M2 NLP
pierre.goncalves2@etu.univ-lorraine.fr

Frédéric Assmus
M2 NLP
frederic.assmus9@etu.univ-lorraine.fr

January 11, 2021

ABSTRACT

This paper is a school project report for a Software Project course. The subject of our project is Question Answering, which we will approach in an applied way, by proposing a system deployed online. This report will cover the different steps that make up this project, from the definition of the subject to the deployment of a Web App. We will address the theoretical and technical aspects of the development of such a system. The final product is a Web App available at this address:

https://ask-me.azurewebsites.net/

Keywords Natural Language Processing · Question Answering · Deep Learning · Wikipedia · BERT

1 Introduction

Many methods in Natural Language Processing (NLP) are related to the comprehension of human language. Recent advances in the field of language modeling have been shown to significantly improve state-of-the-art results on many Natural Language Processing tasks. Since then, a lot of effort has been put towards the development and the training of language models, especially during the past decade. Recently, large-scale pre-trained language models are fine-tuned for each application considered. One of these new language models is called BERT [1], which stands for Bidirectional Encoder Representations from Transformers. BERT is conceptually simple and powerful, it obtains new state-of-the-art results on eleven natural language processing tasks, including Question Answering.

Question Answering (QA) is a NLP subfield that aims to automatically answer questions asked by humans in natural language. An algorithm is given a short text (a context), and is trained to answer natural questions related to that text by highlighting the most relevant passage to answer the question within the context text. Also, there is no doubt that the pre-trained language models are now ubiquitous in Natural Language Processing. However, despite the success of these new pre-trained language models, most available models for Question Answering have been trained on English data, and this makes practical use of such models in all languages except English very limited. On its side, while being the new state-of-the-art, BERT can only be used for answering questions from very short paragraphs.

Question Answering and BERT depend a lot on the quality of corpora, because without documents containing the answer, there is not much that a question answering system can do. It therefore makes sense that larger corpora generally allow for better question-answering performance. Open-domain Question Answering deals with questions about nearly anything. A core goal of these systems is to build a system that can read the web, which is a massive collection, and then answer questions about any topic. These question answering systems could have a big impact on the way that we access information.

With the aim of understanding how the BERT model works and trying to address some problems of the Question Answering task, we wanted to develop a process that would automatically search over a website to find an answer to a

given question. Our aim is, given a question asked by humans in natural language, to understand the meaning of that question and search over a specific website, here Wikipedia, for a context which possibly contains an answer to the question. After that, the context would be given to the BERT model and the answer would be extracted.

2 Subject definition

2.1 Initial subject

Originally, our project was inspired by a Kaggle competition subject [2]. Given a dataset of questions, related Wikipedia pages, and answers, the goal was to train a model that is able to answer users' questions. When given a new question/page pair, the model should identify a short and a long answer to the question, in the linked page. All this work had to be done using TensorFlow 2.0 [3]. As we liked the general idea, we decided to take our inspiration from this subject to come up with our personal subject, which is a variant of it.

2.2 Preliminary thoughts

The subject presented above was very "fine-tuning oriented". We wanted to work on a more user-oriented topic. In the Kaggle topic, the objective was to answer questions from "question - Wikipedia article" pairs. Therefore, this means that the article is known to the person asking the question. Now, how to answer a question that would be asked naively by a user? How to provide an answer to a user whose only material is his question? We thus began to think about a system that would allow us to simply answer questions of general knowledge. We decided to build on the principles presented in the Kaggle topic, and to develop a model based on Wikipedia.

2.3 Actual subject

These thoughts led us to define a project that involves several stages and leads to a final product. The final product we imagined is a system that, from a user's question, provides an answer extracted from Wikipedia. Since we had been inspired by the Kaggle subject, we decided to go for methods close to those proposed for it. So the main idea is to provide meaningful answers without knowing in which Wikipedia article to look for them. A first step would thus be to find out which is the most relevant Wikipedia article that could correspond to a given question. Then, our model would search for the best answer that it could find in this article, and give it back to the user. Our project includes running all of this question-answering process smoothly on a web application. Naturally, this is a summary of the overall goal of the project. In order to get to this point, a large number of steps are required in the processing of the questions.

2.4 Details and tools

The main algorithm of our project will be entirely coded in Python 3.8. For development and testing, we will use Jupyter Notebooks. In a more detailed way, we will need to use several Python libraries for the different processing. Among these libraries are for example SpaCy, NLTK, or wikipediapi. We will talk more about the use of these libraries when the time comes. The language model used for question-answering is BERT [4]. We will use a fine-tuned BERT, trained on the Stanford Question Answering Dataset (SQuAD) [5][6][7]. For the Web App implementation, we will use HTML and CSS, coupled with Flask, which will allow us to integrate our Python scripts to our website. To deploy our solution online, we will use a docker image and Azure ML. All of these technical aspects of our project will be discussed later, in the context of their use.

3 Pipeline

The core process of our project can be seen as a pipeline composed of three main blocks, each taking as input the output of the previous one as described in Figure 1 . The components are :

- First, a question processing system taking in input the user's question and returning the subject of that question.
- Second, a system that finds the Wikipedia page associated with the given subject and returns its contents divided into paragraphs.
- Finally, a system that would run our chosen model, BERT, on each of the paragraphs and return the best answer, which will be our final output.

Each block will be described in the following parts.

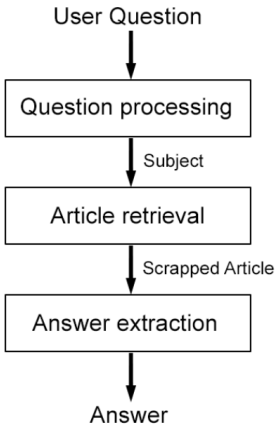

Figure 1: pipeline representation and its components

3.1 Question processing and subject detection

Our system is based on finding a Wikipedia article in which it is possible to identify the answer to a given question. To find this article, it is necessary to identify the subject of the question, or at least its general topic, i.e what the question is about. To do so, we came up with different methods and processes. First, we use Named Entity Recognition (NER). Detecting the named entity avoids ambiguities when linguistic expressions have a meaning in their entirety and are not relevant if we take the words one by one. For example, if we want to find the subject of the question "Who wrote The Lord of the Rings ?" and we don't use NER, our system would end up saying that the part of the question that is most likely to be the topic is "Rings", which is obviously not what we want. Instead, NER will detect "The Lord of the Rings" as a named entity. We consider that if a question contains named entities, then the most relevant topic is likely to be one of these entities. This will depend on the type of named entity. Indeed, there are a number of different types of named entities (see Figure 2).

PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

Figure 2: Different tags for SpaCy's NER

We have arbitrarily chosen which types of named entities are most likely to be question topics. And we decided to keep 9 of them, which we found relevant 'PERSON', 'FAC' ,'ORG', 'GPE', 'LOC', 'PRODUCT', 'EVENT', 'WORK_OF_ART', 'LAW'. We used SpaCy's NER to detect the named entities from these chosen types. If one entity is detected, then we consider it to be the topic of the question. If more than one entity is detected, then we consider the last one to be the topic of the question. If no named entity is recognized, then we apply a different process to the question. This process is the division of the question into noun chunks. Noun chunks are linguistic objects that correspond to sense units. They can be composed of one or more words, as long as this group of words has a proper sense. To do so, we use the noun_chunks function of the SpaCy library. As before, if one chunk is detected, it is considered as the topic of the question. If more than one chunk is detected, we consider the last one to be the topic. If no chunks are detected, we consider that the question has no subject. The chunking process involves some processing details, such as removing interrogative words, which we don't want to be recognized as noun chunks.
To illustrate, Figure 3 is an example of output of this subject extraction component with as input a typical user question, "What is the height of the Eiffel tower". The correct subject, "the Eiffel tower", is extracted.

```
[the height, the eiffel tower]
subject found by noun: the eiffel tower


{'subject': 'the eiffel tower'}
```

Figure 3: output of our subject extraction component with the question "What is the height of the Eiffel Tower ?"

3.2 Article Retrieval

The idea is to retrieve the best Wikipedia article according to the extracted topic. In order to do that we mainly use the Wikipedia API. We found several python libraries managing the use of this API to greatly simplify the process. We used two different ones to build our system because each one offers different functionalities. The first library used is named Wikipedia and allows us to access the Search function, which, given a word, returns a list of existing Wikipedia page names that are close to the query. As show in Figure 3, with the query "the eiffel tower", several article names are retrieved. After selecting the first article name from this list, which is the most correlated to the input word, we need to retrieve the content of the article. The current library gives access to a function to do this but it causes errors and strange behavior.

Indeed, when entering the exact name of a Wikipedia page, sometimes another one is returned. So we used a second Python library called 'wikipediaapi' which does not offer the search function, but can display all the content of a Wikipedia article just by having the name of it as input. So we mixed the use of the two API libraries, using the first to get a relevant article name, and using the second to retrieve the content of the article. The content of the article is then split into paragraphs, ideally following those of the original article. The only reason why our splitting sometimes does not respect the one of the article, is when the paragraph exceeds 500 tokens. Indeed, BERT's input is limited to 500 tokens, so when we face a larger paragraph, we have to split it again. The list of paragraphs is then passed to the following system.

```
['Eiffel Tower',
 'Eiffel Tower replicas and derivatives',
 'List of the 72 names on the Eiffel Tower',
 'Gustave Eiffel',
 'Under the Eiffel Tower',
 'Eiffel Tower (disambiguation)',
 'Eiffel Tower in popular culture',
 'The Man on the Eiffel Tower',
 'Émile Nouguier',
 'Eiffel Tower (Cedar Fair)']
```

Figure 4: output example of the Search Function of wikipedia API with the query "the eiffel tower"

3.3 Answer Extraction

The next step is to process all those paragraphs with the user question in order to retrieve the right answer. We first developed a function named generate_answer to run our selected model. The input is the question associated with the

context text from which the answer must be extracted. Figure 5 shows what the system should do with a question, here "What is the height of the Eiffel Tower": The text highlighted in blue is the answer that will be extracted from the corresponding Wikipedia article, here "Eiffel Tower".

The function performs first a preprocessing, with a tokenization of the two input strings using the tokenizer provided with the model. It then segments the text into tokens and concatenates the two inputs, separated by a token [SEP]. Another [SEP] token is added at the end and a token [CLS] is added at the beginning. This is the format required by the BERT model to represent the structure of its inputs. A forward pass in the model is then performed returning two probability distributions. They assign to each token values that represent the confidence that this token is the beginning or the end of the answer that has to be found in the context text. With a Max function we can retrieve the two tokens that delimit the answer, to reconstruct the whole answer that will be the output of this function. We decided to associate a confidence value to this answer that we calculate by summing the confidence value of our two delimiting tokens in their respective probability distribution.

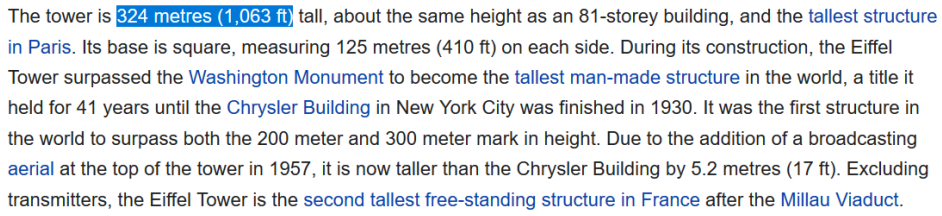

Figure 5: Answer extraction visualization for the question "What is the height of the Eiffel Tower ?"

For the main process, we loop on the paragraphs list previously obtained and run our generate_answer function with the user question and the paragraphs. It results in a list of answers associated with a confidence score for each paragraph. The answer with the best confidence is considered as the most likely to be the best answer. In this process we decide to loop on a limited number of paragraphs for computation time reasons , the forward pass on the model being resource consuming. We have chosen 10 for combining performance and computing time, the first 10 paragraphs of a Wikipedia article being the one where the most important information is.

4 Web application and additional features

4.1 Web application

Our application is a computer-based software program that runs locally on a device. In order to make it quickly accessible to a large number of users, we decided to make it a web application and put our application online. Our web application runs on a web server and allows its user to perform a task over the internet. The purpose of our web application is to let the user access the website at any time so that he can ask his question. We will therefore have to retrieve the question asked by the user in the search bar, then run our system and finally display on the user's screen the answer provided by our system.

For that purpose, we have to create a class named Ask which contains all the functions, from extracting the subject to generating an answer. With that instance, we can build our end-to-end QA system that will follow our process and be able to generate answers to questions. This system will automatically download the BERT SQuAD model if it does not already exist on our system.

We use the Python web framework named Flask which is easy to get started and needs a small amount of code for getting a simple app up and running. Flask does not require particular tools or libraries and allows you to build a web application. It keeps the style of the website consistent and saves a lot of time when creating and updating our application. Furthermore, with this framework, it is much easier to consider deploying our online application without facing major difficulties.

4.2 Deployment with Microsoft Azure

In order to deploy our web application online, there are several solutions. We have chosen one of the most famous: Microsoft Azure. Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers. Before deploying our application, various preliminary tasks have to be carried out. First, we have to create a "requirements.txt" (see Figure 6) file in which we list all the python libraries that have to be installed to run our application without issues.

```
flask==1.1.2
torch==1.6.0
transformers==3.3.1
scikit-learn==0.22.1
nltk==3.4.5
spacy==2.3.2
wikipedia==1.4.0
Wikipedia-API==0.5.4
langdetect==1.0.8
```

Figure 6: requirements.txt used to download the libraries

Second, we need to install the Docker application. It enables us to separate the application from the infrastructure. Docker provides the ability to package and run an application in a loosely isolated environment called a container. With Docker, we build our own image, which is a read-only template with instructions for creating a Docker container. To do so, we create a Dockerfile (see Figure 7) with instructions and a simple syntax to define the steps needed to create the image and run it.

```
FROM python:3.7

WORKDIR /app

# only copy requirements.txt first to leverage Docker cache
COPY /requirements.txt /app

RUN pip --no-cache-dir install -r requirements.txt

COPY . /app

CMD ["python","flask_test.py"]

EXPOSE 80
```

Figure 7: Dockerfile used for our application

After building our image with Docker, we need to push it into our Azure container. Finally, we run this image and deploy it. Azure will automatically create a public URL to give everyone access to our online application. However, deploying our online application comes at a price. We have to create an Azure App Service (a plan) that provides different options and performance according to the budget. Our application needs a minimum of 6GB of memory for a request and the running process takes a maximum of 30 seconds to give an answer with a low price CPU. We opted for the P2V2 or P2V3 (see Figure 8) plan solution in order to prevent our website from crashing if multiple requests are sent in a short period of time.

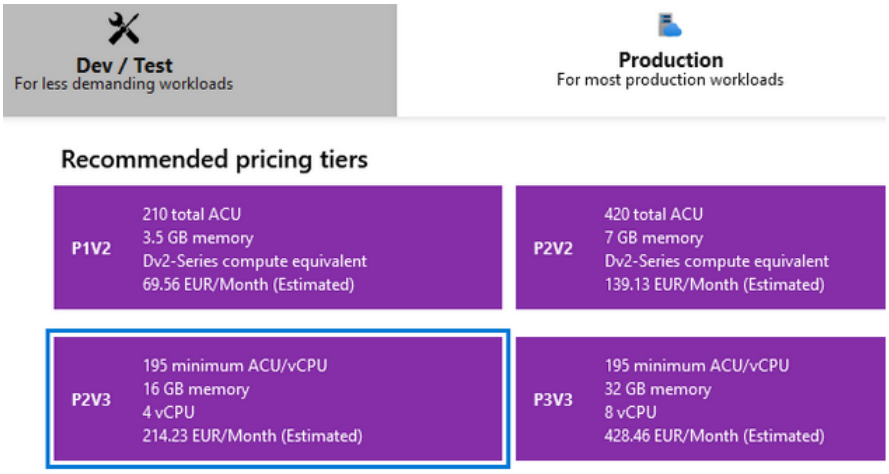

Figure 8: Azure plan with the estimated price

4.3 Making the app enjoyable : multilingual

While testing our application, we noticed that the user was experiencing language problems. We have asked several users of different nationalities to use our application, which was only available in English at the moment. In order to improve our application while making the user experience more enjoyable, we decided to detect the languages used in the question in order to search and give an answer in this given language.

Our system is able to detect 55 languages with the python library “langdetect”. After retrieving the question asked by the user, we detect the language of the question and store it. Then, the language used by Wikipedia libraries is changed in order to search in the detected language. The retrieved paragraphs will then be in the same language as the question and given as input to BERT. The extraction of the answer is done with the same SQuAD-fine-tuned BERT model. Figure 9 10 11 show the question "Who invented the dynamite ?" and its answers in English, French and Dutch.

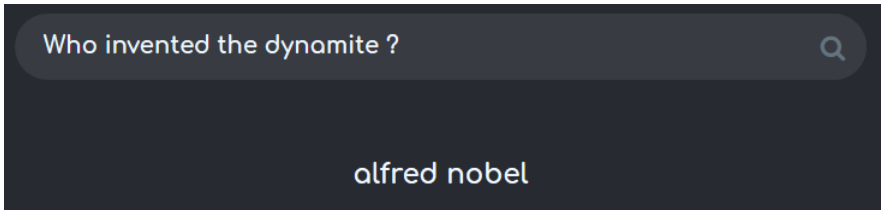

Figure 9: Question in English

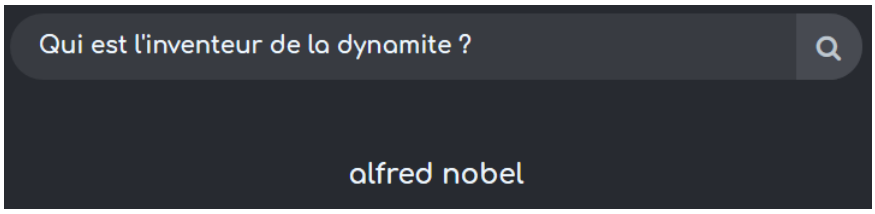

Figure 10: Question in French

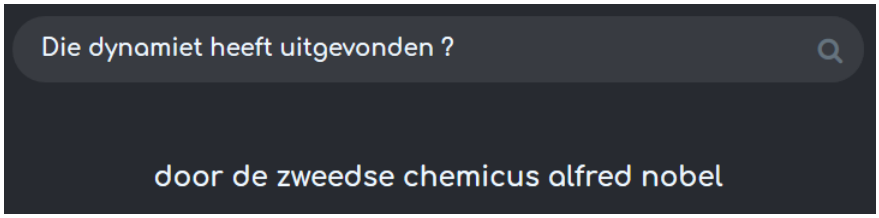

Figure 11: Question in Dutch

We have seen from the previous examples that detection works. However, our QA system is specialised and pre-trained mainly in English. Sometimes the questions are misunderstood and the answers are not satisfactory. In order to improve these answers, it might have been possible to use other models more trained in the detected languages. However, it would have required more resources, there is currently no model that is pre-trained in all languages and they do not perform as well as our model. We have chosen to use only the BERT-large fine-tuned with SQuAD model which is a good compromise to improve our application and user experience.

4.4 Making the app enjoyable : personalized answers

Our system is able to answer general knowledge questions. This does not prevent a naive user from asking a question that is not of this type. What should we do if a user asks a question that does not belong to the range of “answerable” questions ? What should Ask.me output when facing a question which is not a general knowledge one ? The most important thing for us is that the user is not frustrated. Therefore, giving an empty answer or not answering are not options that we consider.

The solution we have found for this problem is to provide personalized answers according to the type of question asked. To do so, we classified questions into 6 categories (see Figure 12)

Question Type	Example
Factual	Where is Barack Obama born ?
Yes/No	Is the earth flat ?
Pseudo Cleft	What she says is true.
Tutorial	How to unlock a closed door ?
Personal	Should I talk to my ex again ?
To App	What's your favorite color ?

Figure 12: Different question types and examples

"Factual questions" are the questions that our system aims to answer. These are general knowledge questions, about information that can be found in Wikipedia most of the time. Yes/No questions are a very common type of question. Unfortunately, our system can't deal with this type of question. Pseudocleft are a type of sentences that start with "what" but are not questions. We extended the definition to other "wh-" interrogative words. We have to detect pseudocleft to avoid taking this kind of sentence as a question. Tutorial questions are the type of question where the user asks for advice or an explanation, which is not the goal of our app.

Personal questions are a type of question that a naive user could ask to have fun or to test the system. We want to detect this type of question so we can show the user that the system understands what he is doing. "To App" questions are another type of "test"/"fun" questions, when the user wants to know things about the engine itself. We want to detect this type of question for the same reason as for Personal questions.

To detect all these types of questions, we wrote some linguistic rules. They go from really basic (e.g. a question is a yes/no question if it starts with an auxiliar) to more complex (e.g. if the head dependency of the interrogative word is an adverbial clause or a subject clause, then it is a pseudocleft). These rules are encoded as functions. Each function corresponds to a type of question and returns whether or not the input question is of this type. The functions are then nested in such a way that there are no "overlapping" problems and that a question type that is one level higher than another does not encompass the whole.

4.5 Making the app enjoyable : additional web features

In order to have a final website that is pleasant to use, some small additions are necessary. First, we have added a "Demo" feature which allows, by clicking on the corresponding button, to automatically write a question (see Figure 13) in the search bar and launch it. This allows a new user to understand in a simple and guided way how to use our application as soon as they arrive on the website, without having to think about a question. This demo is also used to give the user a typical example of a question that can be asked. The following Figure shows an example of a question used for the demonstration. However, if the user is still not sure how to use the engine, they can go to the About Us section and read the description of the app.

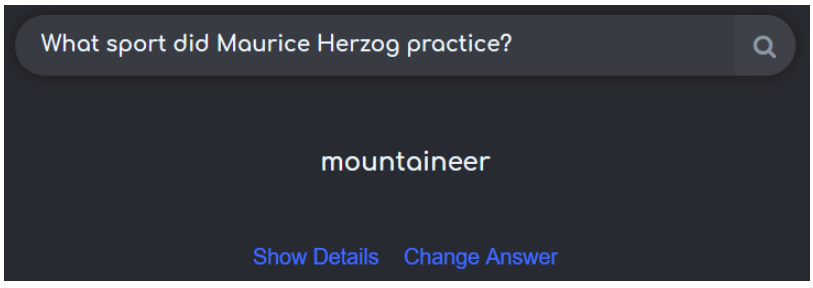

Figure 13: Question used in the demonstration

It is possible that our QA system may make a mistake in choosing the best possible answer to a given question. To solve this issue, we have implemented the "Change answer" feature which, by clicking on the corresponding button, allows the user to access the second and third answer that our model would have displayed instead of the first. In addition, it allows the system to provide several, possibly different, correct answers for a given question. This also makes the use of our application more pleasant.

In addition to the answer we added the possibility for the user to display some context for the answer, with the button Show/Hide Details. The details are the full context paragraphs from where the answer was extracted, associated with the link of the corresponding Wikipedia article (see Figure 14).

Figure 14: Wikipedia context where BERT find the answer

Finally, we have added a loading GIF animation (Figure 15) that appears when our system runs. This allows the user to understand that the query has been taken into account and is being processed. It is necessary because the process can take between 4 to 20 seconds, depending on the number of paragraphs given as input and the resources allocated by the web service.

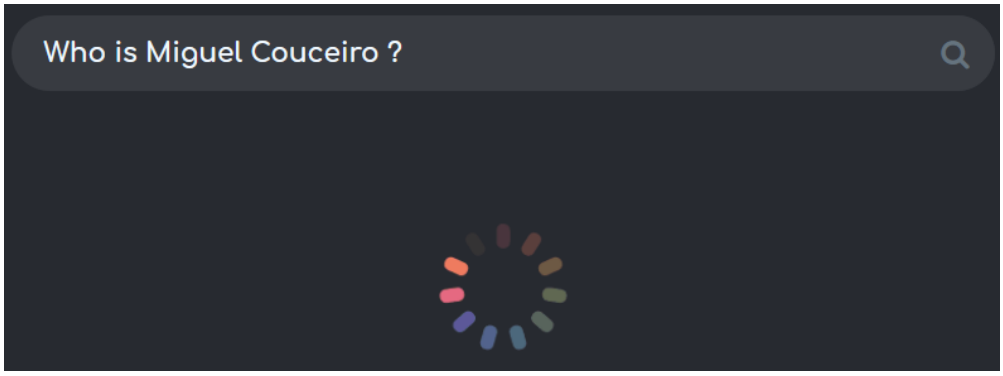

Figure 15: Loading GIF animated

The final result is our algorithm implemented on a Web interface, deployed online, and enhanced with additional features for user experience.

5 Results

In this section we will present the results we obtained with some examples of typical user questions. We will show the system's answers and the final aspect of our working product.

5.1 System result

In this section we show typical questions of different types and their answers. Some have satisfying answers, some have personalized answers, and some have less relevant answers. For those whose answers are less relevant, we will explain the reasons why.

- **Who is Katy Perry ?**
  - an american singer , songwriter , and television judge

- **What is the height of the Eiffel Tower ?**
  - 324 metres (1 , 063 ft)

- **The Black Forest is located in what European country ?**
  - germany

- **How to fix a broken door ?**
  - It seems like you are asking for a tutorial. I can't provide such information to you... maybe you should try to rephrase your question, or ask a more factual one.

- **Who are you ?**
  - Are you trying to get information that concerns me ? I'm shy, so I won't answer, sorry ... Maybe you should try asking some general knowledge questions

- **Who is Dwayne Johnson ?**
  - rocky maivia
  - the rock
  - the son of ata johnson (nee maivia ; born 1948) and former professional wrestler rocky johnson

  Here is the top 3 answers selected by our system: Rocky Maivia being the first stage name of Dwayne Johnson and The Rock his current stage name. The third answer explains who Dawyne Johnson is by talking about his parents. These 3 answers are correct, but none of them seem to be the expected one, which is something like "an actor and professional wrestler". This problem is due to the confidence system that considers the 3 answers above as being the best ones.

- **What is the name of the first american president ?**
  - andrew shepherd

  The answer to this last question is wrong, due to an error in the article retrieval. In this example, the retrieved article is "The American President" which is an American film directed and produced by Rob Reiner in 1995. Due to this error, all the occurrence of the word "president" will refer to "Andrew Shepherd", which is a fictional president in the movie.

5.2 Application result

The final result is the working Web app integrating our system. With the home page (Figure 16) receiving the user's query and displaying the answer and a About us section (Figure 17) presenting the project and the team members.

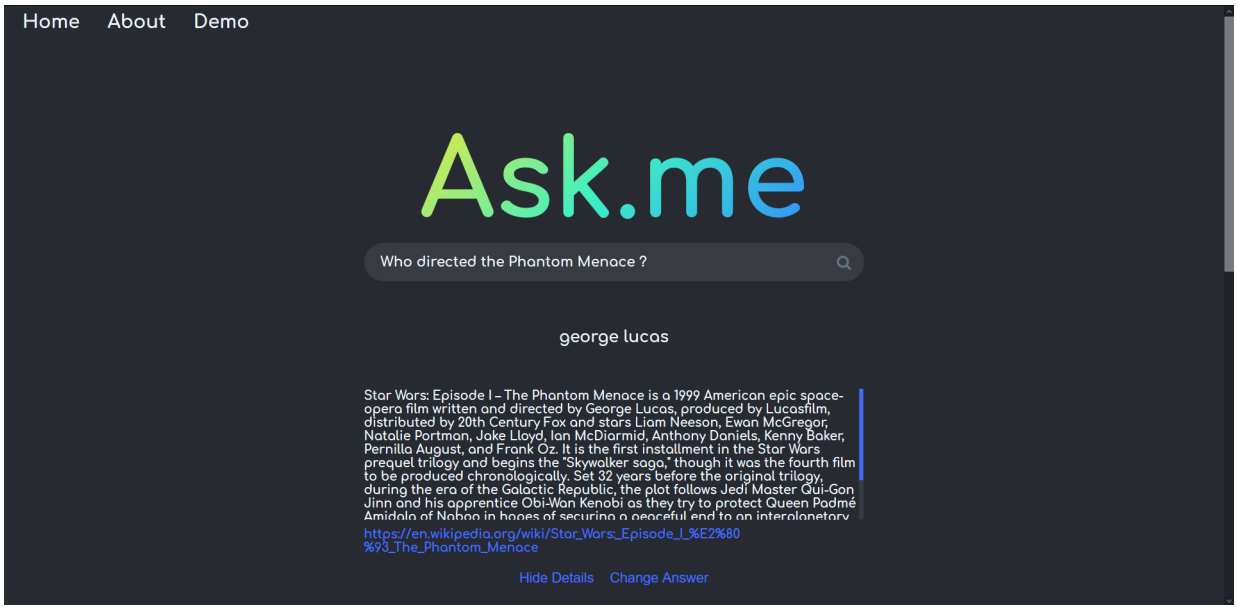

Figure 16: Home page

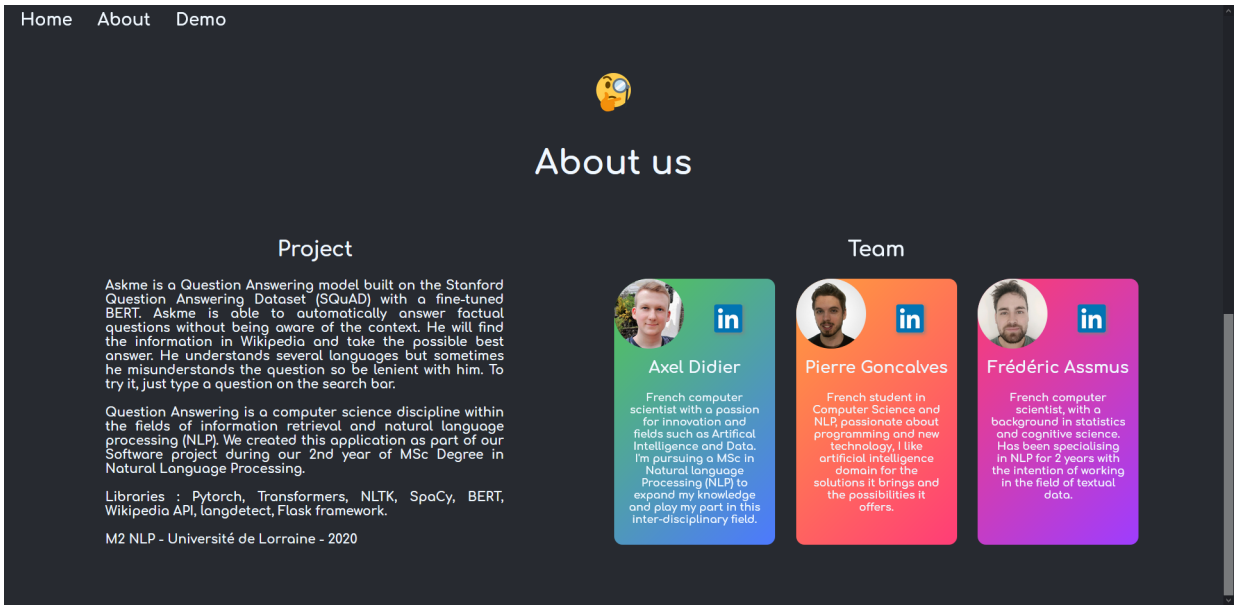

Figure 17: About Us section

6 Conclusion

We wanted to carry out a project around the field of Question-Answering. Our desire to have a concrete project with visible results led us to embark on the creation of a question answering web app.

First, we built a system that - given a question - extracts the subject of the question, finds a Wikipedia article related to it, and extracts a relevant answer thanks to BERT. Then, we created a Web interface for this engine, to which we added various functionalities making it ergonomic: demonstration, detection of the type of question and personalized answer, multilingual support, possibility to change the answer, etc. Finally, we put our application online thanks to the Azure ML tool. The final result is a web app that can be used to answer general knowledge questions efficiently.

We are satisfied with our final product, but it has limitations. These limitations are due to a lack of resources both technically and in terms of time. One example is the number of paragraphs taken into account per Wikipedia article: 15. This number was chosen arbitrarily to have an optimal ratio between the performance and the processing time of the system. A larger number of paragraphs may allow you to find more relevant answers. Unfortunately, the computing power of our tools is not infinite and a good compromise had to be found. Also, the multilingual approach is incomplete. Indeed, we present it as an additional feature. We present it as an additional / "bonus" feature because the results in languages other than English do not come close to English in terms of quality. It allows the user not to be frustrated, to understand the language, and sometimes to provide relevant answers. But we do not claim to have a totally multilingual system.

We had also planned to add relevant features, which we didn't have time to implement. If this project were to be taken further, there are many areas for improvement. For example, we would have liked to implement a disambiguation system. For homonymous words, it would have been interesting if the user could have been offered different meaning options and could choose among them. This would improve subject extraction in ambiguous cases. Also, our system allows answering questions that are general knowledge, but which are open questions. Typically, these questions are those that begin with interrogative words (Wh-questions). But there are some general knowledge questions that can be of another type and whose answer could still be deduced from Wikipedia: polar questions. Polar questions are questions where the expected answer is Yes or No (example: Is La Paz the capital city of Bolivia?). This type of question is very interesting but our system does not allow us to answer them. Indeed, this type of question requires a completely different processing effort, which we did not have time to set up. Such an improvement could be a great step forward for our project.

Overall, we have gone as far - or even further - than we wanted at the start of our project. The results produced by the system are not optimal and could still be improved in many ways thanks to the many tools that evolve every day with the state of the art. However, it is satisfying to use a web app that is online, functional, and meets the expectations we have of it.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] TensorFlow. Tensorflow 2.0 question answering. https://www.kaggle.com/c/tensorflow2-question-answering/overview, 2020. Accessed: 2021-01-11.

[3] Pramod Singh and Avinash Manure. Introduction to tensorflow 2.0. In *Learn TensorFlow 2.0*, pages 1–24. Springer, 2020.

[4] Chris Alberti, Kenton Lee, and Michael Collins. A bert baseline for the natural questions. *arXiv preprint arXiv:1901.08634*, 2019.

[5] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[6] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[7] Zhangning Hu. Question answering on squad with bert. *CS224N Report, Stanford University. Accessed*, pages 12–01, 2019.