École Polytechnique de Montréal

INF6972S : Stage en milieu du travail

Rapport de stage

Maitrise professionelle en Génie Informatique

Option Ingénierie et analytique des données

CLÉMENT BERNARD (2096223)

Directeur d'études :
QUENTIN CAPPART

Responsable de stage :
AURÉLIEN CLUZEAU
Tuteur : AMINE SADEQ

10 Mai 2021 — 27 Août 2021

SUMMARY

This document presents the internship I did at BusterAI, a startup which deals with misinformation on the web. I worked during 4 months with the research team as an NLP (Natural Language Processing) deep learning research intern.

The main goal of the company is to prevent misinformation on the web, which could either be from text inputs to images (like deep-fake). The software which is developed aims to help people to find truth in the amount of information we can find on the internet. The misinformation on the web is mainly about fact-checking : the entailment about a given claim. It means that, given a claim on a subject, a system should find a context (which is an article, a web page, a book, ...) that asserts whether the assumption is validated, refuted, or hasn't enough information to answer.

Therefore, my job was mainly to implement a way to improve the fact-checking method used by the company. I had to implement the results of a recent article about fact-checking brief generator on my own, from the setup of the project to the training and optimization of the model.

keywords : internship, fact-checking, natural language processing, deep learning.

ABSTRACT

Fact-checking has high interest today, as the misinformation spreads very easily on the web. Nevertheless, the amount of websites and information in general is increasing even more, making the task of fact-checking more and more difficult. The aim of my internship was to implement an article which takes claims as inputs, and try to increase the amount of information it contains. It starts by generating questions, where the aim is to split a claim in order to identify the main elements and topics. Then, with the questions and a context, the model should answer those questions. The idea is to add more information to the claim in order to improve the entailment part : give an answer about the veracity of a claim. Then, I had to implement an improvement of the original paper.

During the internship, I also worked quickly on other projects. The FEVEROUS (Fact Extraction and VERification Over Unstructured and Structured information) [1] challenge, which is a recent challenge where the aim is to predict the evidence and verdict for a pair of claims and texts or claims and tables from Wikipedia. I had to implement few elements of the pipeline, which wasn't primordial but could have improved the model.

CONTENTS

CHAPTER 1

BUSTERAI

1.1 Introduction

BusterAI is a software created to 'find truth in the ocean of misinformation'. The main goal of the company is to fact-check the content available on the web by using state-of-the art solutions on both Natural Language Processes (NLP) and Computer Vision (CV).

Fact checking has challenging purposes. It becomes more and more important, but at the same time more and more difficult to counteract. On the one hand, every one can spread false information easily on social media or other platforms with easy access, but on the other hand, checking the truthiness of a fact can take from few minutes until days.

Nevertheless, as the amount of information is increasing, this data can be used to create automatic systems that verify assumptions. This is the idea between artificial intelligence : take advantage of the amount of data to train a model to generalize on this data.

The service offered by BusterAI has three different features :

1. Web portal where one can enter a claim, image or video and the system outputs either the articles that fact-check the given input (with a confidence score) (for the claim) or an overall score of confidence on the veracity of the video/image.
2. A browser extension that summarizes the information of fact-checking.
3. An API to directly query the algorithms.

All the algorithms used are state-of-the-art methods from recent research articles or standard techniques.

1.2 Internship details

My work at BusterAI was to implement a state-of-the art method to improve the models of the company. I was in the research team, which was composed of five full-time research engineers who worked on NLP or CV.

The company is composed of three main teams : the developers, the research team and the product team. I met people from each team, but I only worked with the research team.

I was autonomous in my work. I had guidelines at the beginning, starting from bibliography about the topic of the article, then implementing the paper and after trying to improve what was done.

After a month and a half, the CEO decided to make all the efforts of the company into the FEVEROUS (2.4.7) challenge. Therefore, I had to pause my current work in order to help the team in this race. I worked during a little less than a month with a linguist intern in some experimentations.

Once the FEVEROUS challenge was over (and won by the team), I spent the rest of my time in the project by finishing the reproduction of the results, the improvement part and the setup of the project to make my work reproducible.

The material I had was a Virtual Machine from Scaleway with 16-GB GPU on it and a SSD memory of around 350Gb. I also had a gitlab account, and a Jira and Confluence access to share my progress with the team.

The remote days were free to choose, even if the research team had a dedicated day on Friday. Therefore, I used to go to office on every Friday, and came from time to time on other days during the week (usually not more than twice a week).

I used to publish frequently documents about my progress as Confluence pages, which enabled my supervisor to follow my results. Therefore, what the company keeps from me is a list of documents that explains my work, a git repository with my code and some guidelines to reproduce my results and the weights and datasets used.

CHAPTER 2

FACT-CHECK BRIEF GENERATOR

This project is the implementation of the paper [4] by Facebook AI research. My work consisted of implementing the Question-Answering brief generation, whereas the passage brief and entity brief weren't part of my project. I will explain my work and the paper description in the same time, and compare my results through the sections.

I spent around 1 month and a half trying to understand, implement, optimize the Question-Generation part (2.2), before doing the Question-Answering (2.3) task for 1 month. Then, I had 2 weeks to create a pipeline to integrate my work in a fact-checking application (2.4).

I will start by a quick overview of the paper in 2.1. Then, I will describe the Question-Generation in 2.2, which is how to create questions from a claim. After that, I will detail the method I used to do abstractive Question-Answering in 2.3. Then, I will explain in 2.4 how I decided to use those questions and answers to improve fact-checking.

2.1 Description

2.1.1 Paper

The Generating Fact Checking Briefs paper is an article that tries to improve the efficiency of fact-checking by creating information from claims. The aspect that I kept was the generation of questions and the question answering task. They introduced QABrief-Datsaset, collected by crowdsourcing.

The dataset created contains 6897 claims and 21168 questions with their answers for the training set, whereas the validation and testing set both have 500 claims with 1400 questions and answers.

The QABrieferModel is described in Figure 2.1.

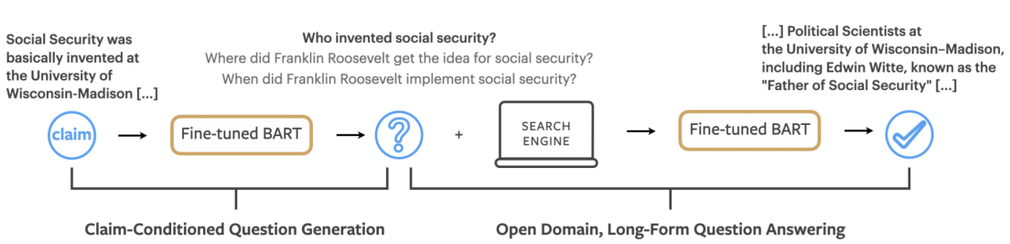


Figure 2.1 – QABrieferModel pipeline

The idea of their work was, given a claim, to highlight the main elements by creating questions. Then, given a search engine and the questions, to answer those queries. With the questions and the answers, they gave it to fact-checkers and observe if it improved the fact-checking or not.

The paper explained that they created the dataset with annotators. They were asked to read a source where a claim was coming from, and then should create questions to decompose the claim. There were other annotators who had to review the quality of the questions, and then other crowd workers had to reformulate the unclear questions. For the Question-Answering, other annotators had the claim, the source, and the questions and could query a search engine to get the answers. Some other validations were done to ensure the quality of the answers.

Therefore, the QABriefDataset is one of the few datasets that deals with both questions and answers simultaneously to combat misinformation. Furthermore, datasets usually rely only on Wikipedia, which makes it hard to rely on in real world use. This dataset aims to be well suited for real case fact-checking, as it allows different sources for the retrieval of documents.

Finally, both Question-Generation and Question-Answering tasks are evaluated with different metrics, which will be explained below. The benefits of the pipeline is evaluated with the accuracy of fact-checking, but with volunteers. They compared the time and efficiency of fact-checking with human annotators, making the results not replicable.

2.2 Question-Generation

2.2.1 Introduction

The Question-Generation (QG) task is a generative problem where the goal is to generate questions from a claim. It should highlight the important elements that compose a claim as well as the structure. In order to have a model that takes as input a sentence, and outputs a sequence of questions, we need to have a sequence-to-sequence model. Nonetheless, the dataset that I had wasn't large enough to train from scratch a model, which required to use a pre-trained model and fine-tuned this one on the given dataset.

The challenge of this task was basically how to create the good outputs/inputs format to enable the model to learn well. The dataset hadn't one claim and one question, which would have been an easy way to implement. It basically had around three questions per claim, which made it harder for the model to learn from.

2.2.2 Models

The paper used BART [6] : Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation and Comprehension. This is a denoising autoencoder for pretraining sequence-to-sequence models. It is composed of a BERT [3] encoder, and a GPT [2] decoder. As it has a BERT architecture, it uses self-attention, which is a method that has recently improved results on NLP tasks on almost every task. There is 10% more parameters on the BART encoder than the BERT one. Indeed, there are two major differences in the encoder compared to BERT : each layer of the decoder performs cross-attention over the final hidden layer of the encoder and BART doesn't use a feed-

forward network before the word-prediction.

BART was trained with corrupted inputs. There are five different noise functions :

- **Masking** : given a list of tokens, it masks random tokens from the sentence. Masking means replacing the given token with a masking token.
- **Permutation** : permute the order of the sentences.
- **Rotation** : select a token, and make it as the beginning of the sequence. It can split a sentence and therefore change the sense.
- **Deletion** : remove random tokens from the sequence.
- **Infilling** : take one or more followed tokens and replace them with a masking token.

BART was trained for different tasks, such as sequence classification, token classification, sequence generation and machine translation. In my work, I was only interested in the sequence generation task, which used the architecture of GPT for the decoder to predict future tokens based on past inputs (autoregressive).

The results of the BART paper showed that, for the sequence generation tasks, BART outperformed almost everywhere the previous results. The tasks used were Summarization (task where the input is a long sequence of text and the output is a summary), Dialogue (task where the model should generate responses conditioned on previous context) and Abstractive QA (given a context and a question, the model should output long answers). The noises they used was a mix of few noises : text infilling, sentence permutation and a masking of 30 % of the tokens.

Another well known model for generative task is T5 [11] : Text-to-Text Transfer Transformer. T5 has an encoder-decoder architecture with layers of self-attention, normalization and feed-forward networks. The model was trained with an unsupervised objective, which means some tokens were randomly dropped, and the model had to predict them. It has been pre-trained on multiple datasets, such as GLUE [14], CNN/DM [9] or SQuAD [12]. It is also possible to do multitasking, which means the model can be fine-tuned for different task in the same time. Therefore, this model was well suited to generative tasks, which made it a good candidate to compare with BART, even if the paper didn't mention T5.

I implemented the code to use one of the following models :

- **BART-base** : a 140M parameters model pre-trained (with 6 layers in the encoder).
- **BART-large** : a 400M parameters model pre-trained (with 12 layers in the encoder).
- **BART-large-xsum** : a 400M parameters model pre-trained on the CNN/Xsum summarization task.
- **T5-base** : a 220 million parameters model pre-trained.
- **T5-large** : a 770 million parameters model pre-trained.

The choice of the models was made by me. I usually used the base models to see if the learning process was running fine before moving to the larger models to increase the performances.

2.2.3 Metrics

In order to evaluate the questions generated by a model, one can use different metrics. Some metrics are commonly used in translation or in summarization. As there is no perfect metric, I had to select the metrics that could be accurate enough to judge the quality of the predictions. Furthermore, as the output is a sequence of strings, the classic machine learning metrics like accuracy, recall, precision or f1 score weren't adequate.

Therefore, I used the following metrics :

1. **BLEU** [10] (Bilingual Evaluation Understudy Score) is an evaluation made for machine translation, but can work for other tasks. Given a candidate and some references (can be more than one), the BLEU score is computed with the following formula :

$$\text{BLEU} = \text{BP}\ exp(\sum_{i=1}^{N} w_n log p_n) \tag{2.1}$$

where BP is described with

$$\text{BP} = \begin{cases} 1 & \text{if c} > \text{r} \\ e^{(\frac{1-r}{c})} & \text{if } c \leq r \end{cases} \tag{2.2}$$

with c the length of the candidate corpus and r of the reference corpus. The geometric average of the modified n-grams is given with the formula :

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n-gram \in C} Count_{clip}(\text{n-gram})}{\sum_{C' \in \{Candidates\}} \sum_{n-gram \in C'} Count(\text{n-gram'})} \tag{2.3}$$

The weights are uniform given by $w_n = \frac{1}{N}$.

This formula computes the precision with multiple references for one candidate, which enables to have a more robust system (with multiple possible translation if this is a translation task). The $count_{clip}$ is a count of the number of common n-grams but without redundant words in the candidate (compared to the words in the references).

The BLEU score can be computed with different values of n-grams, depending on how strict we want to be. For the generation task (QG and QA), I used 4,3,2 and 1 n-grams. The baseline paper for the fact-checking brief generator didn't say explicitly which value for n-grams they used, but I supposed that it was the BLEU 4 score.

2. **ROUGE-L** score [7] : Recall-Oriented Understudy for Gisting Evaluation is a metric commonly used for summarization. It counts the number of common units

(like n-gram, word sequences or word pairs) between a candidate and the references (which should be human texts). There are 4 ROUGE measures : ROUGE-N, ROUGE-L, ROUGE-W and ROUGE-S. I only used the ROUGE-L, which stands for Longest Common Subsequence. It basically counts the longest sequences of words that are common between the candidate and the reference. I only kept the F1-score of the ROUGE-L measure for my problem.

3. **BERTScore** [16] : score based on the similarity of the BERT embeddings of the candidates and references. It is a recent metric that has shown to evaluate well the generation tasks. Nonetheless, I didn't use this metric as the computation time was too high, and the results seemed to give the same hierarchy as the BLEU score. I just wanted to mention it as I worked and implemented the code for it.

To summarize, I evaluated the QG task with BLEU (4, 3, 2 and 1 n-grams) score and f1 ROUGE-L score.

2.2.4 Pre-processing

The QABriefDataset contained, for a given claim and a source (the entity where the claim came from), multiple questions (up to three). Therefore, the paper mentioned three different ways to generate the questions :

1. Generating all the questions based only on the claim.
2. Generating all the questions based on the claim and the source.
3. Generating questions one at a time.

It implied different preparation of the data. I tried six different pre-processing to reproduce the paper strategies and also to try to improve it.

1. **First** : the input is a claim, and the output only the first question. It means we remove the remaining questions. This pre-processing was used to do grid-search to collect the best hyperparameters for the generation.
2. **Concatenate** : we concatenate the questions, separated by a separator token (specific to the model used) for the labels. For BART, the separate token is '**</S>**'. We keep the order that appeared in the dataset, which adds a human bias. For the inputs, we use the claims.
3. **Concatenate-Entity** : we still concatenate the questions, but instead of having only the claims as inputs, we add the source of the claim (which is often an entity like PolitiFact for instance).
4. **Concatenate-one-at-a-time** : this is inspired by the third method of the paper. The idea is to help the model at each time step to create accurate questions based on the claim and the previous questions. Therefore, we start with the source and the claim as inputs, and predict one question. Then, we concatenate the question with the previous inputs, and try to predict the next question. We do this until

there is no more question, and we predict a new token that means the end of the questions. We add this token to both the model and the tokenizer. Nonetheless, this method still has a human bias because the order of the questions we use is random, and doesn't have any sense. If some questions were more general and others more specific, doing this method could help a lot because the model could have started by predicting the general questions before predicting the specific questions. But this is not the case in the dataset.

5. **Basic** : this method was at first an experiment I did. The idea is to train the model with the claim as input, and each question as a separate label. It means, for the same input, the model has different outputs. It doesn't make sense in a mathematical point of view, but the generating methods that can be used enabled this to work quite well. As the model is already pre-trained, it has grammatical knowledge. Then, fine-tuning it with one input and different questions as outputs allows it not to stick into a specific question.

6. **Multihop** : this is an improved method of the Concatenate-one-at-time pre-processing. The aim is to reduce human bias, which makes arbitrary selection for the question that doesn't have any sense. Therefore, this pre-processing adds randomness. First, given the claim, we randomly select one of the question as output. After, we concatenate each question to the claim, and randomly pick a remaining question as label. We increase the number of questions added to the claim at each time step, and either randomly pick a remaining question or the question that is left. When we concatenate all the questions with the claim, the output is a new token that means the end of the questions. Nonetheless, there is still a human bias because we don't consider all the permutations available (for instance, Question A + Question B and Question B + Question A). It remains a bias due to the order we make when we concatenate multiple questions.

A visual summary of the pre-processing methods used is on Figure 2.2.

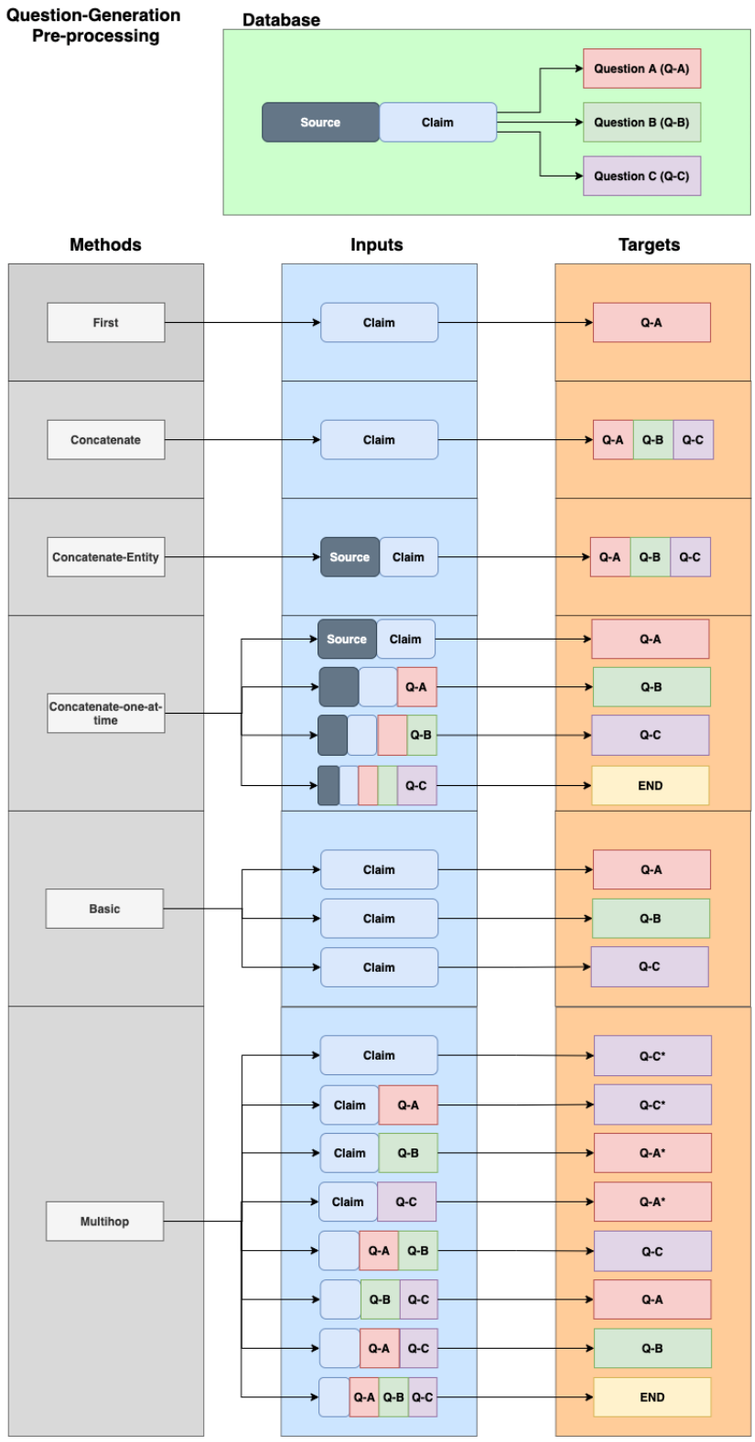


Figure 2.2 – Pre-processing methods

2.2.5 Noises

I reproduced the noise functions used in the BART paper. To do so, I followed the paper explanation, where they described the hyperparameters of the noises.

1. **Masking** : the hyperparameter to use is the percentage of tokens to mask.
2. **Permutation** : boolean hyperparameter. It is either we permute the sentences or not.
3. **Rotation** : boolean hyperparameter, as the permutation.
4. **Deletion** : the hyperparameter is the percentage of tokens to delete.
5. **Infilling** : the hyperparameter is the λ of the Poisson law, which is then going to select a number of tokens to mask randomly according to the law.

To know which noise functions to use in our problem, I tried different values and fine-tuned my model with a simple pre-processing : the **First** pre-processing, which only takes one question as label. The training time was quite fast, which enabled me to try different values. Nonetheless, I didn't use a grid search over different values for each hyperparameter, which would have been a better way to do. The results of the final scores on the validation set are available in Table 2.I.

Metric	BLEU 4	BLEU 3	BLEU 2	BLEU 1	ROUGE-L
Without noise	9.1%	13.5%	19.6%	28.8%	33.8%
Masking (30%)	**11.8%**	**17.1%**	**24.8%**	**38.5%**	**34.3%**
Deletion (20%)	11%	16.8%	24.3%	37.6%	33.5%
Infilling (Poisson law = 3)	9.1%	15.1%	23.2%	36.5%	33.9%
Rotation	10%	15.8%	23.6%	36.5%	33.4%
Permutation	11.5%	16.8%	24.4%	36.9%	32.4%

Table 2.I – Results for noises function on validation set for First pre-processing

It shows that almost every noise function outperformed the baseline (without noise). This is a way to prevent overfitting, and thus to increase the performance of the model. The best noise function is therefore the Masking noise, with 30% of tokens that are masked. It enabled me to know which noise to keep for the training processes.

Nonetheless, I didn't spend a lot of time on the noise effects. I didn't try to use multiple noises, and I didn't use methods to find the best hyperparameters for the noise functions. It would have taken too much time and wasn't worth it in terms of performances. I preferred spending more time on the study of the generation functions.

2.2.6 Generation

Once a model has been fine-tuned on a given generative task, it is possible and advised to use the 'generate' function. The generation enables to have more sophisticated way to create a sequence of words.

I decided to work a little on the different generation functions. The aim was to have the best way to generate questions according to our dataset. Here is a description of the different generation functions studied :

- **Greedy-search** : this is the basic approach to generate a sentence. It takes at each timestamp the word with the highest probability given the previous one. It is easy, straightforward and fast but, it is still greedy and could miss high probabilities hidden behind current low probability.
- **Beam-search** : this is a way to consider more than one word at each timestamp. It keeps track of the *num_beams* most likely combination of sequences, and then choose the new sequence that has the highest probability. It considers further words in a sequence before making a decision, could enable some surprise as human speech, but in the other hand isn't guaranteed to find the output with the highest probability and can suffer from repetition.
- **Sampling** : this is a method not deterministic. It picks the next word randomly, conditioned by its probability with the previous word. In order to reduce incoherency, the *temperature* parameter is here. It reduces the probability of the softmax in order to reduce the incoherent word. If the *temperature* value is set to 0, it is equivalent to have a greedy search. Therefore, reducing the *temperature* value means reducing randomness. Nonetheless, it could create incoherent sequences.
- **Top-k sampling** : this is a method where the most K likely next words and the probability is distributed among those K words. It reduces incoherent output compared to classic sampling method, but limiting the sampling from a fixed K values may produce bad outputs for sharp distribution.
- **Top-p (nucleus) sampling** : this is a method made to prevent the bad words selection enabled by sharp distribution of **Top-k sampling** generation method. Indeed, instead of choosing the K most likely next words, it selects the words where the sum of probabilities exceed the *top_p* value.

For all generation methods, I ran a grid-search over two hyperparameters, and I decided to keep fixed the rest of the hyperparameters (in order not to lose too much time). I kept the hyperparameters that were, for me, the most important to optimize. The metric used to select the final method of generation was the BLEU 4 score, as this was the main metric mentioned in the paper. I used the Concatenate pre-processing, with 30% of tokens that are masked.

The details of hyperparameters fixed and fine-tuned is described in the Table 2.IV.

The results of the grid-search for each method as well as the best hyperparameters are summarized in the Table 2.III.

In order to have more details about the meaning of the hyperparameters, refer to the table 2.IV.

The results show that this is the Beam-search that performed the best, with a BLEU 4 score of 10.2 % for the Concatenation pre-processing. For sake of time, I didn't try with other pre-processing methods as the training time was already quite long. We also see that the non-deterministic methods performed badly, as the training process wasn't made to help randomness (contrary to the Basic pre-processing).

Generation method	Fixed HP	Finetuned HP	Research values
Greedy-search	-	min_length max_length	[1, 2, ..., 10] [40, 50, ..., 130]
Beam search	num_beams (5) early_stopping (True) min_length (10)	max_length length_penalty	[40, 50, ..., 130] [0.5, 1, 1.5, 2]
Sampling	do_sample (True) top_k (0)	max_length temperature	[40, 50, ..., 70] [0.3, 0.4, ..., 0.9]
Top-k sampling	do_sample (True)	max_length top_k	[40, 50, ..., 130] [1, 10, 20, 30, 40, 50]
Top-p (nucleus) sampling	do_sample (True) top_k (0)	max_length top_p	[40, 50, ..., 80] [0.3, 0.4, ..., 0.9]

Table 2.II – Fixed and fine-tuned hyperparameters used in the grid search for the generation functions

Generate method	BLEU 4	Best HP
Greedy-search	9.7%	min_length : 5 max_length : 130
Beam-search	**10.2%**	**max_length** : 130 **length_penalty** : 2
Sampling	7.3 %	max_length : 40 temperature : 0.9
Top-k sampling	8.7 %	max_length : 110 top_k : 50
Top-p (nucleus) sampling	8.8 %	max_length : 80 top_p : 0.9

Table 2.III – Best hyperparameters and BLEU 4 for the different generate functions

Nonetheless, on this part, I didn't try to use T5 with the Basic pre-processing. This arrangement performed better with a mix of sampling as generation function, as used after. I didn't have time to try this and thus to optimize the hyperparameters for this method, as it was a result obtained near the end of the internship.

2.2.7 Training

I trained different models on the different pre-processing available, and I tried to reproduce the scores obtained by the paper. They got a BLEU 4 score between 12.8 % and 13.4 %, where the maximum score they had was when they generate one at-a-time (which should be quite close to the Concatenate-one-at-time pre-processing).

I tried to reduce overfitting with the following methods :

- Increase the dropout on the last layer of the model (up to 30%).
- Freeze the token and positional embeddings for BART (advised on few forums) or freeze just token embedding for T5.
- Implement a learning rate scheduler that periodically decreases or increases the learning rate value.
- Label smoothing (as mentioned in the paper), which avoids the strict label probability.

Hyperparameter	Meaning
min_length	The minimum length to be generated
max_length	The maximum length to be generated
num_beams	Sequence of words to keep in memory at each prediction
early_stopping	Beam search is stopped when at least num_beams sentences finish per batch
length_penalty	Exponential penalty added with the length of the sequence
do_sample	Sampling is activated
top_k	Number of the highest probability vocabulary tokens to keep for the top-k method
temperature	The value used to module the next token probability
top_p	The cumulative probability of parameters. The highest probabilities are kept for nucleus sampling.

Table 2.IV – Explanation of the generate hyperparameters meaning

- Threshold for the gradients

Furthermore, the paper did mention that they used 2048 tokens per batch, which is the maximum allowed tokens by BART-large model. As this is really expensive and not reproducible, I tried with 512 tokens per batch and 180 tokens for the target.

The list of the hyperparameters used are on Table 2.V

Hyperparameter	**Value**
learning rate	3e-5
weight decay	0.01
scheduler	Cosine Annealing LR (t_max = 500 steps, eta_min =3e-6)
batch size	4
max length inputs	512 tokens
max length targets	180 tokens
noise	Masking (30%)
Optimizer	Adam

Table 2.V – Hyperparameters used for the Question-Generation task

Firstly, I spent a lot of time trying to make the model converge, with different hyperparameters values. The learning rate was quite sensitive, and I didn't know for quite a long time if it was the hyperparameter values that prevent the model to converge or the pre-processing of the inputs. I discovered that it was both : the learning rate value should be low, and some pre-processing didn't lead to good results.

In order to have the best results, I used the best hyperparameters for the generation functions found with the First pre-processing with the grid-search (for the beam-search method). For the Basic pre-processing, I didn't use the grid-search to get the best hyper-

parameters, as it was an experiment at the beginning (but revealed to have good results). As the Basic pre-processing had different outputs for the same input, it required adding randomness to generate well, which is what I did (with a mix between top-p and top-k sampling). The list of the best hyperparameters for beam search and sampling are in Table 2.VI.

Table 2.VI – Best hyperparameters for the Generation part

Table 2.VII – Beam search best hyperparameters

Hyperparameter	Value
num_beams	5
early_stopping	True
min_length	10
max_length	130
length_penalty	2

Table 2.VIII – Sampling best hyperparameters

Hyperparameter	Value
num_beams	None
do_sample	True
max_length	80
top_k	50
top_p	0.95
num_return_sequences	4

2.2.8 Results

I summarized all the results I got, taking only the best values and best models I obtained in the validation set. These results are described in table 2.IX.

Model	Loss	BLEU 4	BLEU 3	BLEU 2	BLEU 1	ROUGE-L
Concatenate						
random BART-base	4.88	5.2%	7.4%	10.8%	16.7%	25.1%
BART-base	2.26	**10%**	14.3%	21.1%	32.2 %	35.1%
BART-large	**2.2**	**10%**	**14.8%**	**21.7%**	**32.9%**	**36.1%**
Concatenate-entity						
random BART-base	4.65	5.4%	7.7%	11.2%	17.4%	24.4%
BART-large	**2.24**	**9.6%**	**14.7%**	**21.6%**	**32.7%**	**36.1%**
Concatenate-one-at-time						
random BART-base	5.67	3.4%	5.2%	7.8%	12.1%	22.3%
BART-large	**1.95**	**6.3%**	**10.2%**	**15.9%**	**26.1%**	**28.7 %**
Basic						
BART-base	**2.31**	**6.7%**	**11.1%**	17.8%	29%	27.4%
T5-base	2.79	6.2%	11.4%	**18.8%**	**31.2%**	**31.2%**
Multihop						
random BART-base	3.5	1.4%	2.4%	7%	16%	19.5%
BART-base	**2.98**	**1.5%**	**3%**	**8%**	**16.5%**	**21%**

Table 2.IX – Question-Generation final results

What the scores show is that the Concatenate pre-processing is a good way to generate questions (this is the method that led to the best BLEU 4 score). Adding the source with the Concatenate-entity didn't improve the results (contrary to what the paper said). It could be explained by the fact that, in the training set, around 47% of the examples didn't have source. Then, both the Concatenate-one-at-time and Multihop pre-processing performed badly. All the metrics are low, and we observe overfitting (the training loss is decreasing whereas the validation loss is increasing). The questions generated weren't

precise and usually had grammatical errors. It meant that the pre-processing applied wasn't the good one, and conditioning on the previous questions didn't help the model to generate a new question. This result was different from the paper, which means that my implementation or the way I computed the metrics wasn't right. On the other hand, the Basic pre-processing isn't performing good whereas when we have a look at the generated questions, it seems correct. What is more surprising is that, for the Basic pre-processing, BART-base had the best BLEU 4 score but seemed to perform worst than T5-base. Indeed, I observed that there are a lot of repetitions in the questions generated, which isn't present with the T5-base model. As a matter of fact, the metrics stay an indicator, but aren't perfect tools to judge the results of this task.

Then, I selected the two models I found the best (BART-large with Concatenation as pre-processing and T5-base with Basic as pre-processing). We can see examples of generated questions in the Table 2.X, with an example for either the training, validation or testing sets as well as with a home-made claim.

It shows that neither BART-base nor T5-base (with their respective pre-processing) are perfect. They write questions that have sense, and are most of the time grammatically correct. The questions are always related to the claims, even if they ask different topics than the original labels. Nonetheless, the questions didn't decompose the claim in multiple topics that could increase the information we got from the claim, and we usually see some redundancy in the generated questions.

2.2.9 Issues

I had a lot of struggles to do this training during the beginning of my internship. Indeed, I didn't know if the model didn't learn because of the pre-processing of the data I did was wrong, or if my code wasn't right or if the model just couldn't learn. I tried with different noises, different dropout values and learning rate.

There still remains some issues about the training of the model with Multihop as pre-processing. There are the loss and BLEU 4 score metrics during training that are on Figure 2.3 for the BART-large model with Concatenation as pre-processing. We see that for this example, the loss and BLEU 4 score behaviour make sense : the training loss is decreasing, whereas the validation loss is firstly decreasing before increasing (and overfitting). The BLEU 4 score for both the training and validation sets is increasing, but start being almost constant for the validation set. This kind of behaviour is logical and means that the model is learning to generate questions.

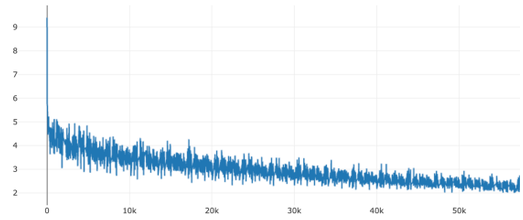
On the other hand, the BLEU 4 score and loss for BART-large with Multihop as pre-processing wasn't doing the same learning process. The metrics during training are shown on Figure 2.4.

The training loss and BLEU 4 score make sense : they are respectively decreasing and increasing, which meant the model learnt to generate the questions with the given inputs. Nonetheless, it was impossible to make the validation loss decreasing : it always

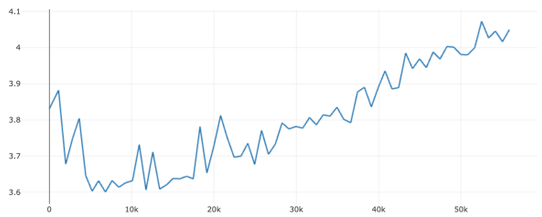
Claim	BART-base (**Concatenation**)	T5-base (**Basic**)	Targets
Training set			
8 Syrians caught at Texas border in Laredo	How many Syrians were caught at the Texas border in Laredo?	How many Syrians have been caught at Texas border?	What were Syrians doing crossing the Texas border?
	What is the source of evidence of the 8 Syrians caught at Texas border?	Where is the Texas border at Laredo?	How were the Syrians caught?
Validation set			
Pre-existing conditions are safe under the Trump administration	How are pre-existing conditions treated under the Trump administration?	Did Donald Trump say the healthcare issue is normal?	What is the Trump administration doing to help people with pre-existing conditions?
	What is the current rate of pre-existing conditions in the United States?	What does the current Obama administration mean when it comes to Pre-existing conditions?	What did Trump do to hurt people with pre-existing conditions?
Testing set			
Obama left 40% of the U.S. with air quality that doesn't meet EPA standards.	How many states have air quality that doesn't meet EPA standards?	Does Obama make sure air quality isn't being bad?	Which presidents have failed to meet EPA standards?
	What is the average air quality in the United States?	Why are many Americans failing to meet air quality standards?	What epa guidlines have presidents failed to adhere to?
Invented claim			
Queen Elizabeth and Duke of Edinburgh receive Covid-19 vaccine	What is the Covid-19 vaccine?	When did Queen Elizabeth and Duke of Edinburgh receive Covid-19 vaccine?	None
	Who is Queen Elizabeth and Duke of Edinburgh?	What is the frequency of Queen Elizabeth & Duke of Edinburgh receiving Covid-19 vaccine?	None

Table 2.X – Generated questions with the two best models and pre-processing

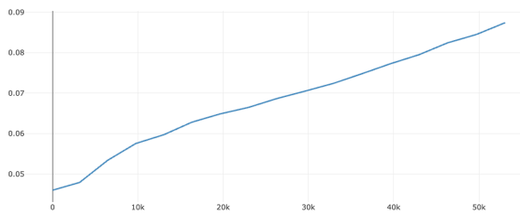
increased (and the BLEU 4 score decreased). This was therefore an overfitting issue, but I wasn't able to prevent this. The model just performed badly. As I had spent quite a long time on this task, I decided to keep the previous pre-processing methods where the training process was doing well and continue with the Question-Answering task.



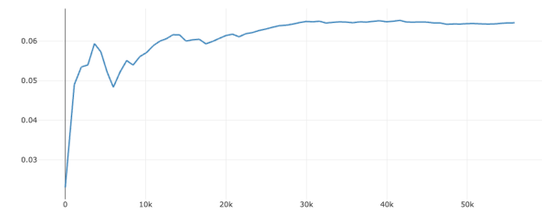
(a) Training loss



(b) Validation loss

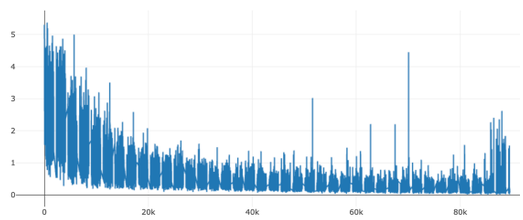


(c) BLEU 4 score on training set

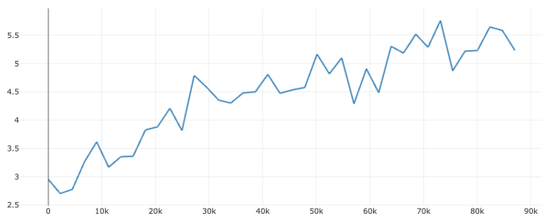


(d) BLEU 4 score on validation set

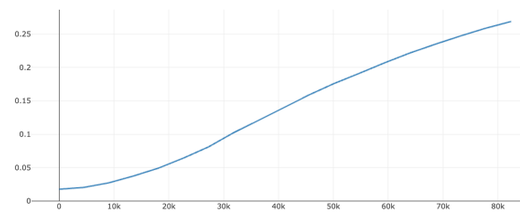
Figure 2.3 – Loss and BLEU 4 score on training and validation set with BART-large with Concatenation as pre-processing



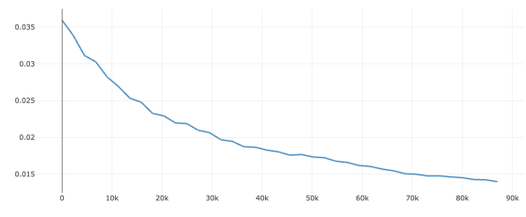
(a) Training loss



(b) Validation loss



(c) BLEU 4 score on training set



(d) BLEU 4 score on validation set

Figure 2.4 – Loss and BLEU 4 score on training and validation set with BART-large with Multihop as pre-processing

2.2.10 Conclusion

As a conclusion for the Question-Generation part, I would say that I tried to reproduce the paper task, but didn't succeed in reproducing the quality of their questions. I didn't know if this was because of the lack of resources (to train on 2048 tokens per batch), the difference in the implementation or the difference in the computation of the scores. On the other hand, I made an architecture where a model can predict multiple questions given a claim as inputs, with grammatically correct questions that have sense. It was still a success for me.

2.3 Question-Answering

2.3.1 Introduction

The Question-Answering (QA) is a task where a model has as inputs a question and a context, and should answer this question. The basic QA task is a span detection : the model should predict the token of beginning and the token of ending of the answer in the context. It implies that the answer is in the context. On the other hand, a generative-QA task uses a model which has as inputs the question and a context, and should generate an answer (that doesn't explicitly exist in the context) from it. This is a different way of thinking the problem, one with a span-based and the other in a sequence-to-sequence base. In my case, I followed the paper's guideline, which says that, as the dataset has answers that can be written by humans from a text, the task should be generative.

The paper mentioned that the dataset is small, which implied that the BART model that was used should be fine-tuned on a different dataset that is bigger. This fine-tuning allows the model to perform better on the given dataset, which is what I tried to do too.

2.3.2 Datasets

The paper explained that it pre-trained the model on the **Natural Question** (NQ) dataset [5].

NQ is a dataset that contains real user google questions and answers found from annotators on Wikipedia. This was created in order to train models on Question-Answering task. It contains 307 372 training examples, 7 830 examples for validation and 7 842 rows for the test set.

As the number of tokens for a whole Wikipedia page is huge, I selected passages where the answer could be. To do so, I used a feature, called *long_answer_candidates* which is a list of passages that could be an answer. Therefore, I concatenated few of these candidates. I did it in order to have less than 1024 tokens per context.

After that, I concatenated the question with the context, and I fed the model with context as inputs, and the answer as output.

For the QABriefDataset, the task wasn't that simple. I had the answer, but not directly the context. I had the URL where the answer came from. Therefore, I had to implement a scrapping method to extract the context from those URLs, and then construct a robust context where the answer should come from. Unfortunately, the paper didn't mention the way they did to construct the context. That's why the method I did was probably not the same as they did, which led to incomparable results.

I extracted the HTML pages with the BeautifulSoop library, and took only the texts (and removed the HTML markers). Then, I had to check whether the answer was inside the context or not. There could be some cases where the answer wasn't exactly in the page (as the annotator can have created the answer from the page). But in this case, I dropped those examples. Thus, as I had too many sentences in the pages, I needed to crop

them in order to create a context where the answer could be in. To do so, I split the text into sentences. I took the answers, and then add randomly few sentences from the text until it has reached a number of tokens (either 512 or 1024 tokens).

To count the number of tokens, I just split the text by one whitespace (which enabled to gain time), which was less accurate than classic tokenization but gave a good idea of the number of tokens.

In order to preserve the order, each sentence that appeared before the answer was appended before the answer, and in the same way for the sentence that appeared after.

In this way, I had around 8500 examples for the training set, and around 700 examples for both the validation and testing set. As the dataset was very small, fine-tuning it in the NQ dataset seemed a good idea.

2.3.3 Metrics

The QA task is basically a problem where we should find an answer to a question. To evaluate the efficiency of the prediction, one should compare the real answer with the generated one. To measure it, I used the following metrics :

1. **BLEU 4** score, as the QG task. It will give an idea about the 4-grams that are repeated.
2. **Exact Match** : a measure that returns 1 if the answer generated is exactly the label answer, and 0 otherwise. It checks if the two strings are equal. I normalized the sentences by lower case them all and removing the multiple white spaces.
3. **F1-score** : the classic F1-score between the target and the predicted answers.

Both the Exact Match and F1-score are used when the QA task is span-based. As in this case I used a generative-based model, the Exact Match metric is a little too strict : generating the exact answer when the answer isn't in the text can be quite hard. That's why I added the BLEU 4 score, which is a good indicator for a generation task.

2.3.4 Training

The paper mentioned that they used a 2048 tokens per batch model, which wasn't something I could afford. Therefore, I thought about a different strategy. I decided to train a single model, BART-base (efficient and not as expensive in terms of resources as BART-large) but with two different values of tokens per batch. I used either 512 or 1024 tokens per batch. The context for the 1024 tokens based context has more sentences than the 512 tokens based context. The difference was due to the creation process, where I explicitly did two versions of the context.

I used the Beam-search generation method, with the same hyperparameters as those used in the QG problem. The hyperparameters are the same as the ones used in the QG task (for BART-base), except that I used 256 tokens for the generated answer.

I trained BART-base on the NQ dataset with either 512 or 1024 tokens per batch, and fine-tuned either the obtained model in the QABriefDataset or the classic BART-base model (not pre-trained on NQ dataset), with also either 512 or 1024 tokens per batch.

2.3.5 Results

The results I got are in the Table 2.XI.

Model	Loss	BLEU 4	EM	F1 score
Natural Question				
Random BART-base (512 tokens)	3.6	8.3%	0%	17%
BART-base (512 tokens)	1.07	41.6%	**15.9%**	43.9%
BART-base (1024 tokens)	**0.76**	**47.4%**	15%	**47.4%**
QABrief Dataset				
Random BART-base (512 tokens)	2.7	11.4%	0%	22.4%
BART-base (512 tokens)	1.27	19%	6%	37.6%
BART-base (NQ) (512 tokens)	**0.76**	**23%**	**7.9%**	**40%**
BART-base (1024 tokens)	0.79	19.9%	6.8%	36.8%
BART-base (NQ) (1024 tokens)	0.8	22%	5.2%	36.6%

Table 2.XI – Question-Answering results on the validation set

It shows that the model learnt well, with a BLEU 4 score that is very high (23%) for the QABrief Dataset. The values for the Natural Question dataset aren't important to focus on, as I just wanted to ensure that it has well been pre-trained on this dataset (which seemed to be the case). The pre-training on the NQ dataset has allowed to have better performances than with a simple fine-tuning on the dataset, which is a behaviour that is in the paper.

The model that led to the best results was a 512-based tokens, which was quite a surprise. I would have thought that as the model has more information, it would be able to produce a more accurate answer. But in reality, it seemed that the shorter the context was, the better was the generated answer. It could be explained by the fact that there was more chance to find arguments in the data as there were less available sentences.

Nonetheless, those results aren't comparable with the paper's results. Indeed, my results were made on a totally different context, as I implemented my own context by concatenated random sentences in the context. Therefore, it makes no sense to compare my results with the paper ones. Still, we can infer that pre-training the model with the NQ dataset led to better results, which was also the case in the paper.

2.3.6 Conclusion

To conclude for the Question-Answering part, I would say that my implementation of this task was controversial. I did use a generative model, with both the question and a context as inputs, and my model predicted an answer with a good score. Nonetheless, the way I generated the context was, for me, experimental and not the best way to do.

2.4 Fact-checking

Once the final QG and QA models were trained, one can generate questions from a claim and then answer them (given the fact we have a way to find a context where the answer should be extracted from). But the paper mentioned that the benefits of these models were to increase the efficiency and reduce the time taken by fact-checker (a human-being) to verify the truthiness of a claim. But they did not use an automatic pipeline to evaluate if the questions and answers improved the fact-checking.

Therefore, I proposed to create a fact-checking pipeline, where the aim was to compare the performances of a baseline model with an improved model with the addition of questions and answers.

2.4.1 Introduction

Fact-checking aims to check the veracity of a claim. It requires finding evidences from a source before being able to infer the relation with the claim. The relation is either SUPPORT, REFUTE or NOT ENOUGH INFO between an evidence and a claim. Therefore, it is a classification with three classes.

As I had around 2 weeks to implement this pipeline, I did some work with the FEVEROUS [1] dataset, which is a fact-checking dataset. Therefore, I decided to use this dataset for checking the results of the fact-check brief generator.

2.4.2 FEVEROUS dataset

FEVEROUS is composed of claims and evidences, where the evidences are Wikipedia passages that should give information on the claim. It can either approve, refute, or doesn't have enough information related to the claim. As the structure of the data for the FEVEROUS challenge is quite complex, I selected only the claims where the evidence came from sentences only. Then, with the good labels, I selected the facts (which is basically all the Wikipedia pages) that are used for the training and validation sets. I extracted the sentences that give the given entailment as evidences and concatenated them.

I simplify the fact-checking task by giving a pair (claim, evidence) as inputs for the model. Indeed, a more difficult task would be not to use the gold evidences, and try to retrieve the evidences from the source. Nonetheless, I didn't work on this part as this is a challenging aspect and can't be done with the time that I had left (2 weeks).

I had 27 975 examples for the training set and 3257 for the validation set. The distribution of the labels are the following :

1. SUPPORTS : 55% for the training set and 43% for the validation set
2. REFUTES : 41% for the training set and 48% for the validation set
3. NOT ENOUGH INFORMATION : 4% for the training set and 9% for the validation set

As we can see, the classes are not equally balanced. The third class, which says that the evidence hasn't enough information to support or refute a claim, has only a small representation. This is usually the case in the fact-checking datasets. I didn't try to work in order to reduce this balance : I just kept the dataset like this, knowing that the data was unbalanced.

2.4.3 Pipeline

In order to compare well the benefit of the fact-check brief generator, I decided to use the same model that was trained to do the classification with and without the questions and answers.

I used RoBERTa [8], which is a BERT model with little modifications (like in the training process or the removing of the next-sentence pretraining objective). It is a well-known model that has proved to perform well on multiple NLP tasks.

I decided to compare four different pipelines :

- **Baseline** : only the claim and evidence as inputs, concatenated with a separator token.
- **Question** : the claim concatenated with questions generated by 3 of the best models (BART-base (Concatenation), BART-large (Concatenation) and T5-base (Basic)) and the evidence as inputs. The idea is to compare if the generated questions add information, even without the answers. An example of the questions generated is in Table 2.XII.

Claim	**Model**	**Questions**
The Golden Calf Occupation Award is one of 16 awards given to film occupations such as cameraman and music composer.	**BART-base** (**Concatenation**)	- What is the Golden Calf Occupation Award? - What awards are given to film occupations such as cameraman and music composer?
	BART-large (**Concatenation**)	- What is the Golden Calf Occupation Award? - What are the 16 awards given to film occupations such as cameraman and music composer?
	T5-base (**Basic**)	- Does Golden Calf Occupation Award affect the career of a cameraman? - What are the qualifications for entering the Golden Calf Occupation Award?

Table 2.XII – Questions generated with the three best models on the FEVEROUS dataset

We observe that the claim is about the Golden Calf Occupation Award, and the generated questions are about what is this the Award, or just reformulate the claim

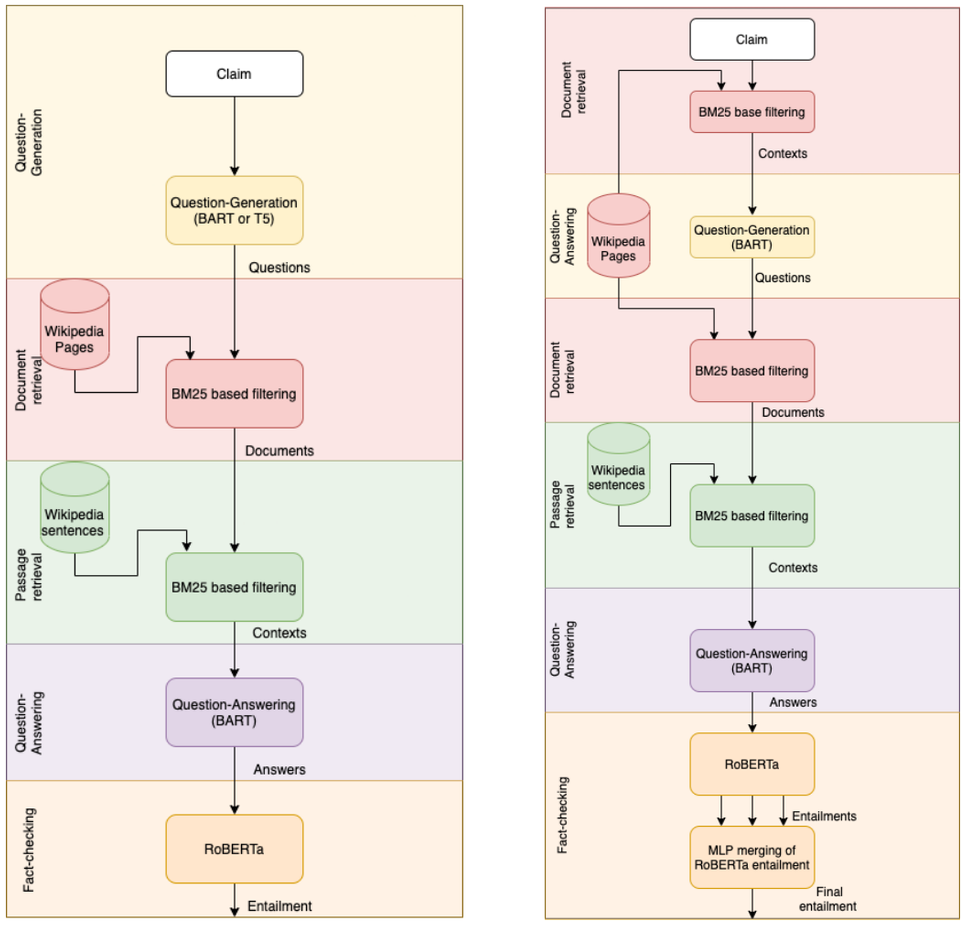
(for the first two models). The last model, **T5-base** (Basic) asked if this award affects the career of a cameraman, which isn't an easy question to generate : it required the model to have a knowledge about what is an award.

- **Question-Answers** : the claim with the questions and the answers associated (with the best model for the QA : BART-base (NQ) with 512 tokens) with the evidence as inputs. One part that hasn't been implemented before was the context retrieval. As the FEVEROUS dataset has a dataset with the Wikipedia pages, I had to retrieve, or at least to construct the context. This is detailed in the section below (2.4.4). The overall pipeline is described in the Figure 2.5a. With the questions and the answers, I concatenated them and feed it into the RoBERTa model for classification. The training is then made in the exact same condition than the previous pipelines, with the same hyperparameters.
- **Improved Question-Answers** : the claims with the questions and answers, but with another way to generate the questions and the answers. It was my mentor who asked me to try a way to improve either the QG model or the QA model. What I tried to improve was the QG task. I tried to change the input of the QG model : instead of using the claim, I decided to use a paragraph. I trained this with the contexts of the QABrief dataset as inputs and the questions as outputs. I used BART-base model, with the same hyperparameters as the BART-base with Concatenation pre-processing. Once the model has been trained, I was able to implement the improved pipeline. It follows the following idea : given a claim, I retrieved the three most related Wikipedia pages. From those pages, the fine-tuned model generated one question for each page. Within those questions, I retrieved the document and then the passages to create a context. With the questions and the contexts, the same QA model as used for the previous pipeline generated the answers. As the concatenation may exceed 512 tokens (for RoBERTa), I added a MLP layer that mapped the (3,512) last hidden layer of RoBERTa (one vector of 512 for each pair of question and answer) into an output of size 3 (for multi-class classification). The idea behind is to enable to take as much information as possible for each pair, and not concatenating them arbitrary. The overall pipeline is in Figure 2.5b.

2.4.4 Document and Passage retrieval

The FEVEROUS dataset has facts that come from Wikipedia pages. These files contain the information needed to do the entailment. I decided to use this as a dataset to get the context from.

First, given a question, I had to select the documents that are candidates to have answers of those questions. To do so, I used Elastic Search (ES), which enabled to store the data and to do the queries easily. I stored the texts of the Wikipedia pages, and compare the questions with all the available pages. In fact, I decided before using Elastic Search to



(a) Fact-check pipeline with fact-check brief generator

(b) Fact-check improved pipeline with fact-check brief generator

Figure 2.5 – Fact-checking pipelines used to improve the baseline

use the BM25 algorithm [13] to compute the top pages related to the questions. Indeed, BM25 is a bag-of-words retriever that can rank documents depending on their contents.

Given a query, which is a question in our case, $Q = q_1, q_2, ..., q_n$, the BM25 score for a document is given by :

$$score(Q,D) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D).(k_1+1)}{f(q_i,D)+k_1.(1-b+b\frac{|D|}{\hat{D}})} \tag{2.4}$$

with $f(q_i,D)$ the frequency of the word q_i in the document, k_1 and b are hyperparameters of default values $k_1 \in [1.2, 2.0]$ and $b = 0.75$, $|D|$ is the length of the document D and $\hat{D}$ is the average document length in the set of document. The IDF (Inverse Document Frequency) is computed as follows :

$$IDF(q_i) = \ln(\frac{N - n(q_i) + 0.5}{n(q_i + 0.5)} + 1) \tag{2.5}$$

with $n(q_i)$ the number of documents containing q_i and N the total number of Wikipedia pages in our dataset.

BM25 has a good trade-off for document retrieval, as it is fast and has relevant results. As I didn't have time to work on a better document retriever, I decided to keep this ranker.

After choosing the method I wanted to use, I decided to use Elastic Search to perform the BM25 ranking. Indeed, the ranking score by default of ES is the Okapi BM25 score, which made it the perfect candidate for my implementation. Therefore, given a question, it retrieved the top documents that maximized the BM25 scores in a short period of time.

In order to create a context, I had to select passages from the top document I got from the BM25 score. To do so, I used also BM25 to rank the sentences of the document, and took the top K sentences. K could have been a hyperparameter to optimize, but for the sake of time, I just used K = 10.

An example of the outputs, associated with the answers from the BART-base (512 tokens) fine-tuned on NQ is in Table 2.XIII. I only kept the questions and the answers, and not included the context because the text was too long. Note that the name of the models are for the generated questions. The QA model isn't mentioned, as this is the same for all.

2.4.5 Metrics

The fact-checking is a three-class classification problem. Therefore, the classic metrics for classification apply. I used the micro-average over all the metrics (except for accuracy). I used the following metrics to assess the quality of my predictions :

1. **Accuracy** : it gives an idea of the number of exact predictions over all the data. But it can be misunderstood if the data is unbalanced.
2. **Precision** : it tells, among the prediction of the class, how many are well classified.
3. **Recall** : it computes how many of the current label examples are well classified.
4. **F1-score** : an average of the precision and the recall. This is a good trade-off for the previous metrics.

2.4.6 Results

I trained the RoBERTa model for each of the previous explained pipelines. The results are shown in Table 2.XIV.

It shows that for the **Question** pipeline, which is just the add of questions, the model outperformed the baseline for the questions from BART-base and BART-large with Concatenation as pre-processing. It means that adding questions add information that can be useful for entailment. But the T5-base (Basic) questions seemed not to outperform the baseline. These questions may not be that pertinent without the according answers.

For the Question Answering pipeline, which is the add of the questions and answers, only the model with the T5-base questions and answers outperform the baseline. It means that, with the answers, the information could improve the entailment to fact-check. More surprising, the BART-base questions, associated with the answers, didn't outperform the baseline. It could be caused by the limited number of tokens of RoBERTa (512 tokens), and the fact that I concatenated the questions and the answers with both the claim and

	BART-base (**Concatenate**)	T5-base (**Basic**)
Claim	Neotysonia is a genus of plants in Gnaphalieae tribe within the Asteraceae family that is endemic to Asia and has only one known species, Neotysonia phyllostegia.	Same
Questions	- What is Neotysonia phyllostegia ? - What is the genus of plants in Gnaphalieae tribe within the Asteraceae family ?	- Which other species is related to Neotysonia ? - Who is the tribe who is in asteraceae in Asia ?
Answers	- Neotysonia is a Genus, genus of Australia, Australian plants in Gnaphalieae, pussy's-toes tribe within the Asteraceae, daisy family. - Micropus, the cotton-seeds, is a Genus, genus of flowering plants in the Gnaphalieae, pussy's-toes tribe within the Asteraceae, daisy family.	- Neotysonia is a Genus, genus of Australia, Australian plants in Gnaphalieae, pussy's-toes tribe within the Asteraceae, daisy family. - Nabalus Nabalus is a Genus, genus of Asia, Asian, and North_America, North American flowering plants in the Cichorieae, dandelion tribe within the Asteraceae, daisy family.

Table 2.XIII – Questions and answers generated for a claim with the FEVEROUS dataset.

Model	Loss	Accuracy	Recall	Precision	F1-score
Baseline					
Baseline	0.52	83 %	61%	55.8%	58%
Question					
BART-base (QG)	0.53	83.4%	61.2%	55.6%	58%
BART-large (QG)	**0.53**	**83.9%**	**61.5%**	**55.9%**	**58.6%**
T5-base (QG)	0.53	82.5%	60.6%	54.9%	57.6%
Question-Answering					
BART-base (QG)	0.53	82.5%	60%	55.2%	58%
T5-base (QG)	**0.5**	**83.8%**	**61.5%**	**55.9%**	**58.6%**
Improved Question-Answering					
BART-base (QG)	0.57	81.6%	59.8%	54.4%	56.9%

Table 2.XIV – Scores on the validation set of FEVEROUS for the entailment.

the gold evidence (which could exceed the available token number). It could also be explained by the bad quality of the questions, as they seemed to just paraphrased the claim. Thus, the T5-base model which seemed to have more pertinent questions wasn't performing well with only the questions because the answers weren't available. Once the answers are present, it could add relevant information, whereas the first two models could have answers with known information, explaining the bad performances. The explication is still unsure and this is just my reflexion of this behaviour.

Finally, the improved method didn't perform well. It is the least performant model, whereas it was thought to be the best. I didn't have time to search why exactly it happened. My thoughts were that the QG model, trained on the contexts of the QABrief dataset, is biased by the context of the dataset. Indeed, the conception of the contexts isn't perfect and can be easily better implemented. Thus, the questions generated should not be accurate and so the associated answers.

2.4.7 Conclusion

The fact-check brief generator achieved to increase the overall fact-checking task with either only the questions or with the answers. It outperformed the baseline, but still wasn't brilliant. A real-world use of fact-checking doesn't have gold evidence with a given claim. Therefore, it would have been great to take time to see if the questions and answers could help to retrieve the evidence in the Wikipedia pages. I tried very quickly this during the end of my internship, but it wasn't performant.

Furthermore, the dataset used was restricted (only to the Wikipedia pages), and very strict because it took only data where the evidence are sentences. This was just to experiment a fact-checking task and to see if my QG and QA works could have applications. The results aren't significant, but are interesting considering the application was very specific.

The fact-checking using information added by generative models remains very interesting, and I would have spent more time on this if I could. There remains a lot of different ways to incorporate questions and answers for fact-checking, or in other NLP

tasks. The generative tasks can be useful if it is well-used, especially if this is associated with non-generative tasks that lack of out-of-the box data.

CHAPTER 3

FEVEROUS

FEVEROUS (Fact Extraction and VERification Over Unstructured and Structured information) is a challenge where claims are annotated from Wikipedia pages (which could be either sentences, cells or tables) that says whether the claim is supported, refuted, or doesn't have enough information to take a decision.

All the research team worked on this challenge, which led to win the first place at the official leaderboard. I'm not going to describe neither the pipeline of the solution nor the full description of the challenge as it was a subproject for my internship.

I worked with two linguistic interns in order to give some insights of the statistics of the dataset, which I will quickly explain on the section I. Then, my work was to enhance the BM25 ranker to increase the Wikipedia pages prediction with deep learning methods.

3.1 Description

The FEVEROUS challenge is composed of 87 026 verified claims which are associated with Wikipedia pages. Each page can either accept, refute, or hasn't enough information for a given claim. Each claim as evidence, which is either a sentence, a Wikipedia cell or both. The goal is therefore to retrieve the evidence given a claim. The authors of the paper even constructed a baseline that extracts the evidences (with a different model for the cell selection, as it isn't as simple as sentences) and then entail with a RoBERTa model [8]. I won't describe all the pipeline of the baseline, as this isn't directly related to my work.

3.2 Statistics

My workmates and I worked a little about the statistics of the dataset. We tried to describe some specific features of the data with some interesting statistics or distributions. The aim was also to help the research team in order to perform well for the challenge. Therefore, we tried to find elements that could differ from the training to the validation set, which could also be a difference between the training set to the testing set as the validation and the testing set share the same distribution. Nonetheless, we didn't find a lot of important differences through our researches.

As all the statistics we got didn't lead to interesting results, I won't describe all of them. I'm just going to describe some of the statistics we studied at and a few interesting results.

The important statistics are described in Table 2.XII. The main difference we saw between the training set and validation set is that the number of cells per table per claim for the validation set is higher by more than 1 unit than the validation set. Nonetheless, that difference wasn't exploited, as it isn't significant enough.

Information	Average value (training/validation)
Number of sentences per claim	(2.21, 2.12)
Number of tables per claim	(1.06, 1.07)
Number of cells per table per claim	(5.97, 4.9)
Sentence length with punctuation	(27.54, 28.14)
Sentence length without punctuation	(24.23, 23.88)
Number of document per claim	(4.9, 4.25)
Number of distinct document per claim	(1.2, 1.2)

Table 3.I – Statistics for the FEVEROUS dataset

We also did some distributions of the features. The list of the distributions are the following :

- Distribution of sentences per document per claim
- Distribution of tables per document per claim
- Distribution of cells per document per claim
- Distribution of cells per table
- Distribution of labels

One interesting distribution is on Figure 3.1. The distribution was the same for the validation set : for instance, the proportion of supported labels was higher for the datasets with only sentences. On the other hand, the mix of sentences and tables as evidence led to more often refuted entailment, for both training and validation set.

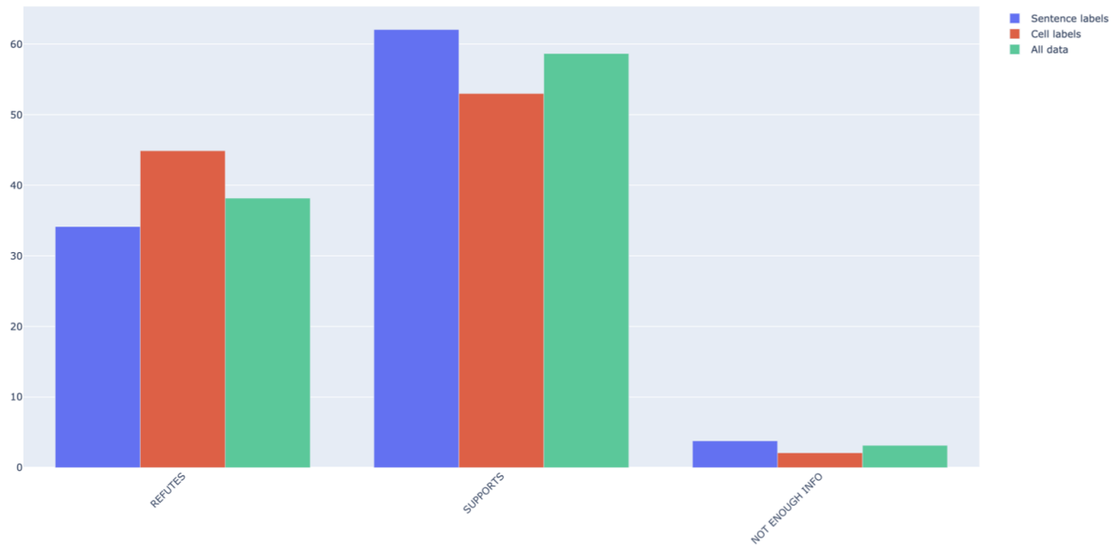


Figure 3.1 – Percentage of labels depending on the evidence type (green : both, blue : sentences, green : tables) for training set

This work was first to look at the differences between training and validation set, then to try to output some outliers or interesting information. But we didn't find such information with the amount of time we had. Finally, this work was also to manipulate the dataset in order to understand well the problem and then to start doing more precise work.

3.3 BM25 re-ranking

One of the task of the FEVEROUS dataset was the Document Retrieval. The aim is to find the documents that are more likely to contain the evidences (which could be either sentences, tables or a mix of both). The research team used a BM25 algorithm to rank the Wikipedia pages, and it worked pretty well. Nonetheless, I had to try to improve this baseline by creating a re-ranker model that re-rank the BM25 results.

3.3.1 Re-ranker

The basic re-ranker I implemented was a binary classifier with either BERT [3] or RoBERTa [8] for sequence classification. I used both the gold labels (real pages where the evidences are) and the results of BM25.

For a given *claim* with gold pages A and B, and pages predicted by BM25 C and D, the inputs and labels for the algorithm was the following :

Inputs	Labels
$claim + A$	1
$claim + B$	1
$claim + C$	0
$claim + D$	0

Table 3.II – Inputs and outputs of the re-ranker

We observe that I penalized the false predictions of BM25, whereas the true labels were put forward.

Nonetheless, BM25 predicted the 200 best pages, which meant there were too false labels. I took only the false pages from the top 15, and I repeated the examples with true labels in order to have the same proportion of both classes.

After that, once the model was trained, the idea was to sort the pages from BM25 and re-rank them according to the probability given by the trained model.

3.3.2 MAP : Mean Average Precision

To judge the result of the re-ranker, the research team and I used the MAP (Mean Average Precision) score over the top K predictions. It gives and idea about how well the good pages are classified over the top K predictions.

The MAP is defined with the following formula (for n classes) :

$$MAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k \tag{3.1}$$

with AP_k the Average Precision over the given class. As in our case there were only two classes (either the prediction was on the top K or not), the formula is given by :

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k) \tag{3.2}$$

with n the number of thresholds. This is a way to summarize the precision-recall curve into a single value. In our case, we considered as well classified a page when it was on the given top K predictions. We denoted with MAP@K this score.

3.3.3 Multihop Re-ranker

I tried another way to improve the re-ranker. Indeed, the classic re-ranker put equally the weights on every page, and didn't reward the model for predicting all the gold pages (there are multiple pages as evidences).

In order to take into account the combination of pages, I rewarded the model when it predicted well the pages when all the gold pages were present. To do so, given a *claim* with gold pages A and B, and pages predicted by BM25 C and D, the inputs and labels for the algorithm was the following :

Inputs	Labels
$claim + A$	0
$claim + B$	0
$claim + A + B$	1
$claim + C$	0
$claim + D$	0
$claim + C + D$	0

Table 3.III – Inputs and outputs of the multihop re-ranker

The model should therefore make more weights on all the true pages, and not only one. But it required thinking differently the inference part : we can't feed the model with a page title and observe the score of confidence for the label one. As the model was trained to predict the class one when the inputs had the claim and all the true pages, it could be difficult for him to do the same when it has only one page as inputs. Therefore, I worked on different inference strategies.

3.3.4 Inference

For the Multihop model, it required a more well-made technique to take advantage of the training process. I thought of three different inference techniques :

- **Beam search** (BS): take the top K pages, and then create all the permutation of tuples. Then, I took the tuple that maximized the score, and predict them as evidence pages for the given claim.
- **Beam search average** (BS avg.) : take the top K pages, and compute the score for each permutation of tuples. Then, I computed the score for each page by averaging the scores obtained whenever this page appeared in the tuple of the created permutation. It allowed to have an idea of how well the page performed in all the different situation. Then, I sorted the pages predicted by BM25 by this score.

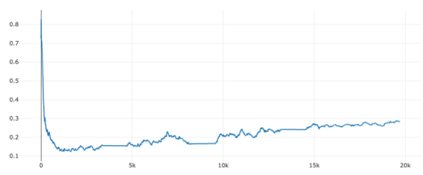
- **Beam search concatenate** (BS conc.) : take the top K pages, and compute the page that has the higher score. Then, concatenate it with each page and compute the score. If the score is higher to the previous score with only the first page, I replaced the page by the two pages, and kept continuing the process. Otherwise, I kept the page as prediction.

I tried all of these methods as inference for the Multihop re-ranker, and select the best method with the MAP score.

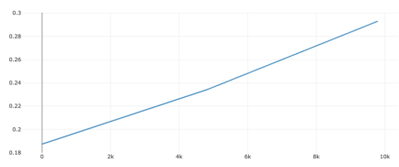
3.3.5 Training and results

As the classification is a binary classification, the classic metrics for binary classification apply : accuracy, precision, recall and F1 score.

I trained either BERT or RoBERTa on binary classification with the explained inputs and outputs for both classic re-ranker and Multihop re-ranker. An example of the loss function for the both the training and validation set is available on Figure 3.2 for the classic re-ranker. We observe that the model had struggled to learn the task. Indeed, the training loss first decreased and then increased. On the other hand, the validation loss is just increasing, which meant the model could overfit. Nonetheless, the behaviour of the training loss doesn't look like an overfitting. It seemed that the model wasn't able to learn the task (maybe the complexity of the model wasn't high enough to learn well). That's what make us think about an improved model : the Multihop re-ranker, that gave more rewards on the combination of all the gold pages.



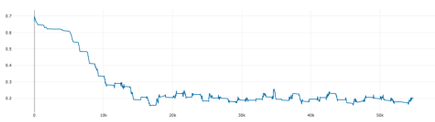
(a) RoBERTa training loss for re-ranker



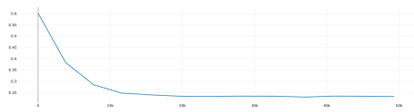
(b) RoBERTa validation loss for re-ranker

Figure 3.2 – RoBERTa loss for the classic re-ranker

An example of the training and validation loss for the RoBERTa model trained with the Multihop data is on Figure 3.3. It shows that the training process was doing fine : the training loss is decreasing as the validation loss. The task seemed more simple and maybe more clear for the model to learn from.



(a) RoBERTa training loss for Multihop re-ranker



(b) RoBERTa validation loss for Multihop re-ranker

Figure 3.3 – RoBERTa loss for the Multihop re-ranker

The final metrics on the validation set for both classic re-ranker and Multihop re-ranker are described in Table 3.IV. It showed for all the re-ranking task, it was RoBERTa (large) fine-tuned that performed the best (contrary to BERT). What was more surprising is that the metrics and so F1 score were quite high for the re-ranking task. It meant that the model did learn well, but very quickly, which could explain the training curve behaviour. The scores for the Multihop re-ranker are quite good too, with around 88.9 % of F1 score. It didn't overfit and learnt well the task.

	Accuracy	**F1 score**	**Precision**	**Recall**
Re-ranker				
Random BERT	56.3%	33.2 %	30.1%	40.4%
BERT	88.2%	86.4%	88%	87.9%
Random RoBERTa	59.5%	33.7%	29.7%	41.8%
RoBERTa	**92.5%**	**90.7%**	**91.7%**	**91.9%**
Multihop re-ranker				
Random BERT	31%	24.8%	23.5%	51%
BERT	89%	88.1%	88%	91.2%
Random RoBERTa	29.6%	22.7%	14.8%	50%
RoBERTa	**91.5%**	**88.9%**	**88.8%**	**92.7%**

Table 3.IV – Metrics of the best models for each re-ranking model on the validation BM25 predictions

Let's now have a look at the most important metric : the MAP score. The results of the MAP scores for all the previous models are described on Table 3.V.

	MAP@1	MAP@3	MAP@5	MAP@10	MAP@15
BM25					
Baseline	80%	89.9%	91.6%	93.2%	94%
Re-ranker					
BERT	78.4%	88.4%	90.8%	93%	94%
RoBERTa	**85%**	**92%**	**93%**	**93.6%**	**94%**
Multihop re-ranker					
BERT	83.5%	91.7%	92.5%	93.5%	94%
BERT (BS)	78.9%	81.4%	81.5%	81.5 %	81.5%
BERT (BS avg.)	84.%	91.7%	92.5%	92.5%	92.5%
BERT (BS conc.)	83%	90.7%	91.3%	91.3%	91.3%
RoBERTa (BS avg.)	**87.3%**	**92.8%**	**93.3%**	**93.3%**	**93.3%**

Table 3.V – MAP scores for the re-ranking models

The MAP scores showed that both RoBERTa re-ranker and Multihop re-ranker outperformed the classic prediction of BM25 (respectively up to 5% and 7.3% for the MAP@1). Note that the Beam Search inference methods were very time-consuming to perform with RoBERTa-large. Therefore, I tried it with BERT, which was faster to execute. I took the best inference method from BERT, which was a little better than the BM25 but not better than the re-ranking method with RoBERTa. Then, I tried with the fine-tuned RoBERTa and obtained the best performances.

There are two main conclusions of the previous results : weighting the pages together led to better results than just using one page (comparison between the classic re-ranker

and Multihop re-ranker). Secondly, training a binary model from both the gold labels and predictions from BM25 led to outperform BM25 predictions, which was the aim.

3.3.6 Conclusion

The work I did for the FEVEROUS aimed to explore experimentations that weren't crucial for the problem, but could help a little to improve the performances of the whole pipeline of the research team. It enabled me to understand and use the FEVEROUS dataset, which helped me after for the fact-checking part. The results were good, but at the end weren't used in the final pipeline.

CHAPTER 4

CONCLUSION

My work at BusterAI enabled me to deep into Natural Language Processing. I was able to develop my Python skills, to explore some real-world projects in company. I had the opportunity to discuss with researchers about recent papers or some code details.

4.1 Question-Generation

The work I did on the Question-Generation task was a little controversial. I felt frustrated as I didn't succeed to reproduce the exact performances of the paper, and struggled a lot to improve the models. I had models that generate questions, but they didn't split the claim into important facts as we wanted to. I did a lot of experimentations for this part that I didn't mention in my report : I tried to generate questions with other datasets and other models. I found the results weren't worth mentioning it.

4.2 Question-Answering

This part was hard to reproduce, as the paper didn't explain in detail the creation of the context. Therefore, my results aren't comparable with the paper results. I just had the metrics and few examples to observe the results, which seemed to make sense. Using a larger dataset to pre-trained was useful and helped the model to perform better. The generative task isn't easy, as the generation part is something that differs from the training aspect.

4.3 Fact-checking

This work was done during the last two weeks of my internship, and was made in a rush. Therefore, there are a lot of things I would have wanted to do better and more properly. I strongly wanted to create a whole pipeline where the QG and QA results could be explicitly exploited. This pipeline was all made in my own, from the thinking of the dataset to the execution (contrary to the QG and QA where I tried to follow the paper guidelines). The dataset used was specific to evidences that are only sentences from Wikipedia. It was therefore more an experience than a real world pipeline. All the results weren't logical, which made me perplex, as I didn't have time to dig into this in more details. But at the end I succeeded to make a clean report and gitlab, to implement all the pipelines even if all the results weren't expected.

4.4 FEVEROUS

As all the research team was focused on this challenge, I had to explore this dataset and its distinctive features. I firstly try to have an overall idea about the general statistics

and distributions of the dataset. Then, I had to do more specific work by trying to improve the Document Retriever model. It was a way to leave my current internship subject and take some distance. Therefore, I enjoyed that part because it was very challenging, I could exchange more with the team and discover different subjects.

4.5 Personnal thoughts

I'm happy to this internship. I was very interested in the NLP domain, and this internship helped me to discover even more. I was very autonomous, which was a good thing because I could either read papers or try to code to improve my work. The atmosphere was wonderful and convivial, and I felt welcomed since the beginning. The technical aspect was challenging, and I learnt a lot.

4.6 Acknowledgment

I would like to thank all the people of BusterAI, from my daily advisors Amine and Mostafa which always answered my questions with delight, to my manager Aurélien who taught me some programming and mentalism tricks. That was with pleasure that I worked with the linguist teams Célia and Sanda, and with humour that we discussed with Thanh, Patrick and Camille. The discussion with the CEO Julien was very rewarding, and I thank him for accepting me in this internship. I enjoyed talking about foods with Cedra, Léa and Hugo, who advised well on the good plans at Paris. I regret not being able to talk a little more with the Development team and the newcomers of Buster. All in all, the experience I had was very pleasant, and the atmosphere was motivating. I learnt a lot, had excellent colleagues and I had pleasure going to work.

BIBLIOGRAPHY

[1] Aly, Rami, Guo, Zhijiang, Schlichtkrull, Michael Sejr, Thorne, James, Vlachos, Andreas, Christodoulopoulos, Christos, Cocarascu, Oana, Mittal, and Arpit. Feverous: Fact extraction and verification over unstructured and structured information. 2021.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[3] 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.

[4] Angela Fan, Aleksandra Piktus, Fabio Petroni, Guillaume Wenzek, Marzieh Saeidi, Andreas Vlachos, Antoine Bordes, and Sebastian Riedel. Generating fact checking briefs. 2020.

[5] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. 2018.

[6] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[7] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. 2004.

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692v1*, 2019.

[9] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çaglar Gulçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. 2016.

[10] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. 2002.

[11] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

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

[13] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. 2009.

[14] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *ICLR 2019*, 2019.

[15] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771v5*, 2020.

[16] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675v3*, 2020.

Appendix I

Implementation details

For the implementation part, I used Pytorch Lightning with MLflow to track the training curves. I used the HuggingFace [15] models for BART and T5. As the BLEU and ROUGE scores weren't implemented in the torch metrics library, I implemented it myself (a month before the end of the internship, pytorch released a version with the implementation of these two metrics). I used flake8, pyenv and black to clean my code and the dependencies.

I had as resource a Virtual Machine with a 16Gb GPU and a SSD memory of around 300Gb. As the HuggingFace library enabled to output the loss when we feed the inputs and the targets, I implemented the code to either use my own cross entropy loss or use the one available by the library.

For the preparation of the datasets (for instance with Natural Questions or FEVEROUS), I mostly used Jupyter notebooks.

I used unit tests for most of the functions or classes I implemented. For instance, for each noise function for the Question-Generation problem, I implemented a unit test with a given input and a hard-coded output, and each time I did a merge request, the tests were done to check if the code was running well.

I also implemented some BDD (Behaviour Driven Development) tests, which is a way to make the code comprehensible by everyone with plain texts. I did it at the beginning of the project in order to learn how to do it, but I didn't have the time to do it at the end of the project.

I made available all the training and evaluation commands in a Makefile. The hyperparameters were all in one python file, which it was easy to modify.

In order to load a given model, I decided to change the path manually in the Makefile, where I had a global variable for each different model used (for the different tasks). I could have added a line in the hyperparameter file, but I didn't, as I considered that it wasn't related directly to this file.

The weights of the models were stored in AWS (Amazon Simple Storage Service), such as the datasets (both the original and the modified ones). I made a script to load the weights and datasets in order for the research team to reproduce my results.

I used a docker instance to launch Elastic Search.

I used a library made by the company to integrate my code with their habits. The library was basically classes to simplify the setup and CLI of the training process. Therefore, I usually inherited those classes and then overwrote the needed functions.

I tried to make my code as clean as possible. To help to do that, I used enumeration classes. It helped me to explicit the different possibilities for variables clean in a file. For instance, no one knows all the pre-training models available for BART. Therefore, I created a class that explicitly listed all the available models (that I used). I did it for the

kind of models I used (RoBERTa, BART, T5, etc) or even for the translation of the class into a label (SUPPORT label for fact-checking is the class 0, etc) and for other useful variables.

I wrote README files : one to explain my whole pipeline of all the internship, and other ones specific for each task that gives the information to reproduce the results.