SAPIENTINO
A BERT-BASED, OPEN DOMAIN QUESTION ANSWERING SYSTEM

Gabriele Barreca, Mario Bonsembiante and Gemma Martini
University of Pisa

Abstract

Question answering (QA) systems can be seen as information retrieval systems which aim is to respond to queries, stated in natural language, by returning short answers or long sentences. The "so-called" *open domain QA task* adds the challenge of understanding if the answer to the selected question may or may not be found in a given paragraph, which content has been buried within large text corpora, such as Wikipedia.

Building such systems for practical applications has historically been quite challenging and involved. The spectrum of possible answers given a question and a paragraph, moves from the "simple" *yes/no answers* to the longer and more articulated *long answers*, to then get to a trade-off between expressive power and succinctness, the "so-called" *short answers*, which aim to enclose the answer in a single and possibly short sentence.

In this paper, we present a BERT-based implementation that solves an open domain QA task, providing all the three categories of answers listed above, with particular attention on the most widely studied kind, i.e. short answers. We achieve pretty good results, although not as good as the state-of-the-art, that was not the purpose of this work.

As expected and already stated in previous work, we conclude that predicting long answers per se is pretty unreliable, while much better results are achieved if the short answer is predicted and then enlarged with the whole paragraph it lies in, from the original text.

Contents

1 Introduction

In this paper we present **Sapientino**, a full pipeline for answering to open domain questions. We developed in Flask an API for answering general open domain questions. We then built a website and an app that utilizes this API for answering user defined, open domain questions. By means of our platform, any user can ask questions and **Sapientino** uses Wikipedia pages and our BERT-based model for answering these questions. This API is easily transferable to new implementations, e.g. a vocal assistant.

This report is structured as follows: in Section 2, the overall architecture is described and then (Section 3) we dig into the details of the choice of NQ dataset. In Section 4 we explain the architecture of **Sapientino**, followed by a study of the performances and a mention (Section 5) to what is yet to come.

2 The architecture

Here we present the general architecture of our API. The server is implemented using Flask, a famous micro web-framework written in Python.

The application we designed has been developed in Flutter, a tool for UI interfaces development, launched by Google that allows to use a single codebase for iOS, Android, web and so on and so forth. The appearance of **Sapientino** is displayed in Figure 1.

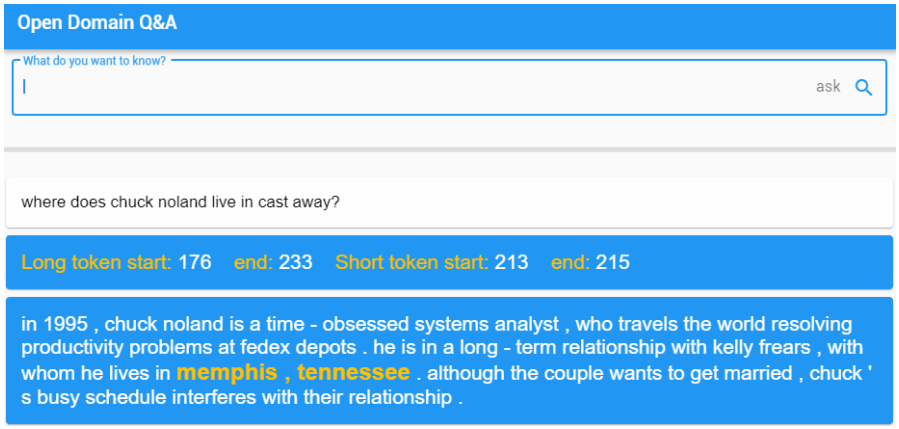


Figure 1: User interface of **Sapientino**.

The pipeline is straightforward (see Figure 2): given a question we start by prepossessing it. We remove the stop-words with the *spacy* library and we add to the query the keyword *Wikipedia*. We feed this bag of words to the Google search engine for finding the fittest Wikipedia page[1].

[1]At the moment we do not use any spell or grammar checker for the question. However this is an interesting feature to add in the future in order to make the bot more robust to misspelling.

Once such page has been found, it is pre-processed by means of *beautiful soup* library: in particular we keep only the main text of the page and we discard all the HTML attributes and some tags (but not all of them). Eventually, we feed the question and the processed Wikipedia page to our BERT-based neural network that predicts the start and end span of the answer.

In this report, we will not dig deep into the first two steps of the pipeline, since they are well known to the reader, while we will discuss how we chose the dataset and how we build the BERT-based model.

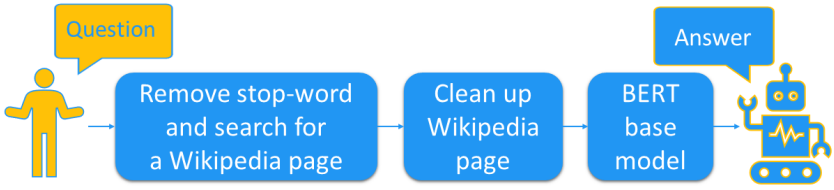


Figure 2: The full functioning of **Sapientino**.

3 Dataset

There are many different datasets that are available for the open domain question answering task. Among all, one of the most famous is the SQuAD dataset, available in two versions: v1.1[1] and v2.0[2]. However, these two datasets suffer from observation bias, because the questions are provided *only after* the human annotators have read the given passages. For the implementation of **Sapientino** , we decided to use the Natural Question (NQ) [3] dataset which is not affected by this bias. The questions present in NQ dataset are sampled from Google search engine's queries made by users and then filtered by some handcrafted rules. The remaining questions are those that are likely to be answered based on a Wikipedia page.

IN NQ, given a query, one or many Wikipedia pages are associated to it. In addition, two different types of answers are annotated and have to be provided: a short answer and a long one which corresponds to a macro section that encapsulates the short one.

'The inclusion of real user questions, and the requirement that solutions should read an entire page to find the answer, cause NQ to be a more realistic and challenging task than prior QA datasets.' [3]

The complete NQ dataset measures 42GB[2] and it contains all the HTML of the Wikipedia pages. It goes without saying that such a huge quantity of data may be difficult to handle.

The solution proposed by the authors makes use of a simplified version of the datasets taken from Kaggle competition, that discards from the origial dataset some

[2]Compressed

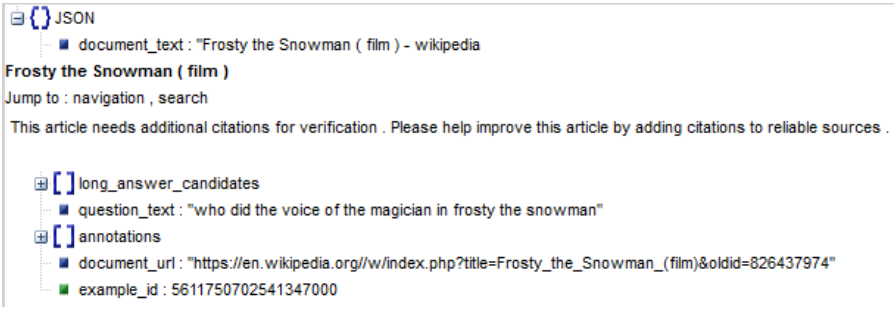


Figure 3: Broad structure of an input pattern.

insignificant HTML. This simplified dataset provides a much smaller training set (4GB[3]) and a little test set ($\approx$ 17.9MB).

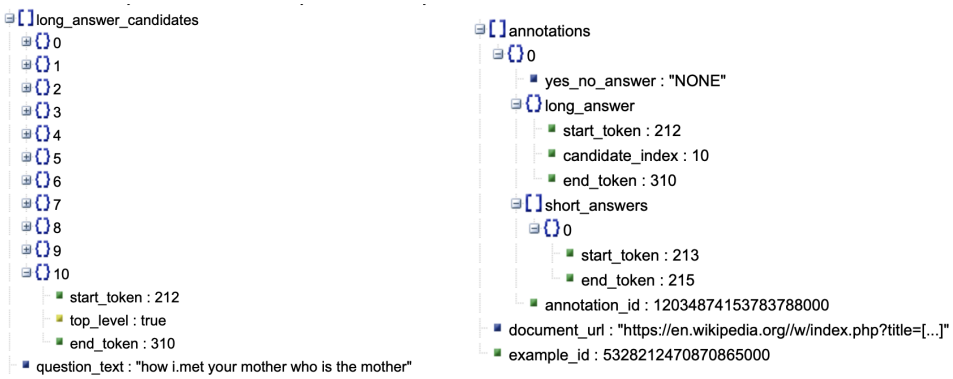
On average, input patterns in the NQ training set have a size of 10MB and the whole training set is stored in a `.jsonl` file of size $\approx$ 17 GB. It goes without saying that loading both BERT's checkpoints and such file into RAM is not possible, so we managed to overcome this problem by splitting the file into chunks with a size smaller than 100MB. Each training pattern is a `json` object, it is stored in a line and it has the following structure (see Figure 3):

- `document_text`: the HTML (cleaned of some tags) of the paragraph that may contain the answer;
- `long_answer candidates`: contains the original question and a list of start and end positions of candidates for the answer (an example in Figure 4a);
- `annotations`: contains three sub-objects, that represent if the question allows a "yes-no" answer, the information about the short answer and the information about the long answer respectively (as shown in Figure 4b). It is possible that a question does not allow to be answered looking at the paragraph given as input. In that case the fields in the `long_answer` object have value -1 and the list `short_answers` is empty.

  In total, annotators identify a long answer for 49% of the examples, and short answer spans or a yes/no answer for 36% of the examples. We consider the choice of whether or not to answer a question a core part of the question answering task, and do not discard the remaining 51% that have no answer labeled.[4]

As a conclusion, concerning the test set, we can say that is composed by 346 items but, unlike the training set, we only have the `long_answer candidates` and therefore we do not have information about the short answers.

[3]Compressed



(a) `long_answer candidates`. **(b)** `annotations`.

Figure 4: More details about the fields of an input pattern.

4 Model

In this section, we introduce BERT (the transformer on which we based the implementation of **Sapientino**) and the two different neural network architectures that we designed for fine-tuning.

4.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) [5] has been introduced by Google in 2018 and it has been defined as the biggest leap forward in the past five years. BERT has led to impressive gains in many natural language processing tasks, ranging from sentence classification to question answering.

Bert is a deep bidirectional encoder which is pretrained on a huge corpus like Wikipedia for learning contextual representations. It is then finetuned for solving different task and it is the current state of the art of many NLP tasks.

A common approach used during the Kaggle competition in order to improve the performance is adding new tokens. In particular the tokens related to the main HTML tags, namely `'Dd'`, `'Dl'`, `'Dt'`, `'H1'`, `'H2'`, `'H3'`, `'Li'`, `'Ol'`, `'P'`, `'Table'`, `'Td'`, `'Th'`, `'Tr'`, `'Ul'`. This should help the classifier in fact more than 90% of the start token of the long answer is one of the tag above. For this reason we decided to add these tokens as well.

After the introduction of BERT in 2018 many upgraded version were released in the last few years. Some studies carried out by Y. Liu et.al. showed that BERT model was *significantly undertrained*[6] and performed a more effective hyper-parameter tuning, giving birth to RoBERTa.

In September 2019, the work of Lan et.al., lead to the development of a lightweight version of BERT(ALBERT [7]), that allows two parameter reduction techniques to lower memory consumption, increase the training speed with respect to BERT and achieve better scaling performances.

4.2 BERT for question answering

We decided to tackle the open domain QA task by creating a stack of two neural networks, forming a two-layer architecture, as shown in Figure 5.

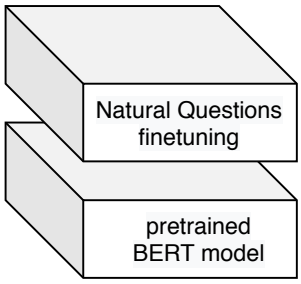


Figure 5: Sketch of the architecture of **Sapientino**.

The first layer is built using BERT's [5] checkpoints [4] from Hugging Face, while the second layer is a neural network that uses BERT's embeddings and the Natural Questions (NQ) [4] dataset [5] with the aim of obtaining the answer to the question.
In the following paragraphs, the reader can find a more detailed explanation of the two layers.

As done by Alberti et.al. in [8] we add on top of the BERT model a fully connected neural network layer which predicts the start token and the end token of the short answers. BERT (and other transformers models like ALBERT) outputs a sequence of contextualized token representations $\mathbf{H^L} = [\mathbf{h_1^L}, \mathbf{h_2^L}, \ldots, \mathbf{h_T^L}]$

$$(h_1^L, \ldots, h_T^L) = BERT(x_1, \ldots, x_T) \tag{1}$$

$BERT_{BASE}$ consists of $L = 12$ transformer layers, each of the ones uses 12 heads for the attention mechanism, where $h_t^L \in \mathbb{R}^{768}$. When using BERT for open-domain question answering it is common practice to introduce some special markup tokens, namely `[SEP]` and `[CLS]`. In particular, we build a crop with the following structure (as done in [7])

```
[CLS]question[SEP]wikipedia page[SEP]
```

where the maximum length of the crop above is 512 tokens. In general, a Wikipedia page is not contained in 512 tokens and this is why we needed to split such page into multiple crops. We will focus on this issue in Section 4.3.
In **Sapientino** we add on top of BERT three dense layers followed by a softmax activation function, i.e.

[4] In practice, we run experiments using also ALBERT [7]. In the future also RoBERTa's [6] checkpoints will be used.

[5] Some qualities of NQ are the following: (1) the questions were formulated by people out of genuine curiosity or out of need for an answer to complete another task, (2) the questions were formulated by people before they had seen the document that might contain the answer, (3) the documents in which the answer is to be found are much longer than the documents used in some of the existing question answering challenges.

- ◇ The first layer takes the $\mathbf{H^L}$ and predicts the start of the short answer (denoted as l_s),
- ◇ the second layer predicts the end of the short answer (denoted as l_e)
- ◇ and the third one predicts the start of the long answer (denoted as l_l).
- ◇ A categorical cross-entropy loss function is used in order to fine-tune the model.

This approach is inspired by [9][6] and it is slightly different from what was proposed by Alberti et al.. They use a common approach for this task, i.e. to predict *only* the short answer and then identify the bounds of the containing HTML tags.

Conversely, we already stated that our solution aims at identifying both the short and long answer; namely, given the output embedding $\mathbf{H^L}$ we predict $l_s = W_s\mathbf{H^L}$, $l_e = W_e\mathbf{H^L}$ and $l_l = W_l\mathbf{H^L}$. For what concerns the crops that *do not contain* the answer, the model learns to predict the `[CLS]` token both as start and end token.

Formally, the loss function is computed as

$$\mathcal{L} = -\frac{1}{3}\left(\sum_t^T \mathbb{1}(\mathbf{s}^t)\log \mathbf{l}_s^t + \sum_t^T \mathbb{1}(\mathbf{e}^t)\log \mathbf{l}_e^t + \sum_t^T \mathbb{1}(\mathbf{l}^t)\log \mathbf{l}_l^t\right) \tag{2}$$

where $\mathbb{1}(\mathbf{s}^t)$, $\mathbb{1}(\mathbf{e}^t)$ and $\mathbb{1}(\mathbf{l}^t)$ are 1-hot representations of the three target variables we decided to predict, namely start and end boundaries of the short answer and the starting point of the long answer.

4.3 Preprocessing

In order to train the BERT base model we have to apply some preprocessing. As [8] and [10] we accomodate BERT pretrained input size constrained of 512 tokens by splitting larger sentence into multiple spans over the Wikipedia article using sliding windows. Different stride could be used, we used a stride value of 256.
We have to build the crops with the structure described in section 4.1, where the corpus is obtained with the different sliding windows. Another consequence of splitting each Wikipedia article into multiple spans is that most spans of the article do not contain the correct short answer (only 65% of the questions are answerable by a short span and, of these, 90% contain a single correct answer span in the article with an average span length of only 4 words). As a result, there is a severe imbalance in the number of positive to negative (i.e. no answer) spans of text. The authors of [8] address the imbalance during training by sub-sampling negative instances at a rate of 2%. On our baseline we emulate this sub-sampling behavior when generating example spans for answerable questions we keep the unanswerable crops with probability 3%.
We decided also to address this problem with a novel approach. We mask the start-end token losses when the span is unanswerable. As consequence of this we train the start-end span classifier only to predict answerable questions. We then used another classifier on top of the [CLS] token in order to predict whether or not the crop contain an answer to the question or not or not.

[6] This team reached the second position in the Kaggle competition with BERT large.

All the above prepossessing steps are based on the code of the second classified of the Kaggle competition [9].

4.4 Our implementation

We propose another implementation of the model, an ideal improvement of the model described in the last subsection. In this improved implementation, the problem of dealing with unbalanced data (see Section 4.3) is mitigated by the use of another dense layer with sigmoid activation function. Such layer, hereafter called *answerable layer* decides if a question is answerable or not. Furthermore, we use this information to compute the loss as follows:

$$\mathcal{L} = \begin{cases} l_{\text{start}} + l_{\text{end}} + l_{\text{long}} & \text{if answerable} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where l_{start} (l_{end}) is the categorical cross-entropy between the start (end) position of the short answer in the target and the predicted start (end) position; l_{long} is the categorical cross-entropy between the start position of the true answer and the position of the guess in the long answer.

As concern the use of the new predicted Answerable value, we use this information also in the evaluation step. Trivially, before to check if an answer exceeds the threshold to be considered correct, we check if Answerable value is greater than 0.5, if yes we go on with the computing otherwise we considered the question unanswerable.

4.5 Hyper-parameters selection

We used ADAM as optimizer as common practise with BERT based models. We did hyper-parameter search only on the learning rate while for other parameters we took values from [8] or [9]. We can see on table 1 the value we tested for the learning rate and the β_1, β_2 we used with ADAM. in order to test the different learning rates we trained the model for one epoch with these different values.

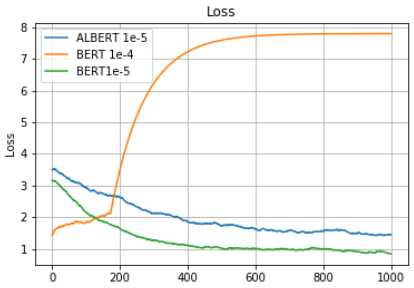
Parameter	Values
η ("learning rate")	$1e-4$, $1e-5$, $1e-6$
β_1	0.9
β_2	0.999
batch size	8 ($\text{ALBERT}_{\text{BASE}}$), 4 ($\text{BERT}_{\text{BASE}}$)

Table 1: Hyper-parameters values.

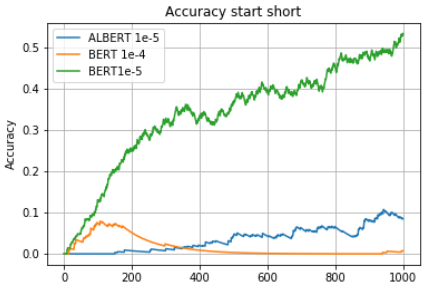
We can see in Figure 6 the results with BERT with learning rate $1e-5, 1e-4$ and ALBERT with learning rate $1e-5$ (this was also selected, however we omit other plots for simplicity)[7]. We can see clearly that BERT needs with a batch size of 4 a learning

[7]We used partial accuracy as metric instead of classical accuracy. Partial accuracy takes into account only the accuracy when the crops is actually answerable and omits on the accuracy the unanswerable questions.

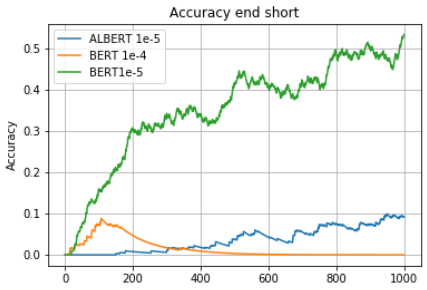
rate smaller than $1e-4$ otherwise it will not converge. For value smaller than $1e-5$ the convergence is slower. On the other hand even the best ALBERT model we had performs far worst than BERT. For this reason we decided to discard ALBERT and work only on BERT.
We have selected the learning rate also for the second model proposed and it is, not surprisingly, equal to $1e-5$ as well. We can see now on fig. 7 the partial accuracy reached by the two models. The model proposed by us clearly outperforms the original model with respect to the partial accuracy, that is when answer an answerable question. This is not surprising because this model mask the loss when the crops have no answer ans learn only to answer answerable questions. We can see also on fig. 7d the accuracy reached by the binary classifier that predict whether the question in answerable or not with the crops. We had an accuracy of 89%.



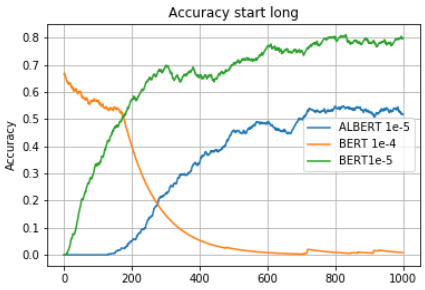
(a) Loss (mean of categorical cross-entropy values of start position, end position and long start position) values over the number of processed crops.



(b) Partial accuracy of start position values over the number of processed crops.



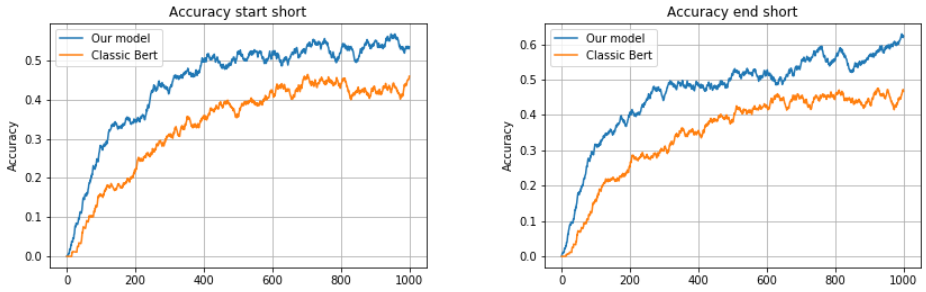
(c) Partial accuracy of end position values over the number of processed crops.



(d) Partial accuracy of start position values of long answers over the number of processed crops.

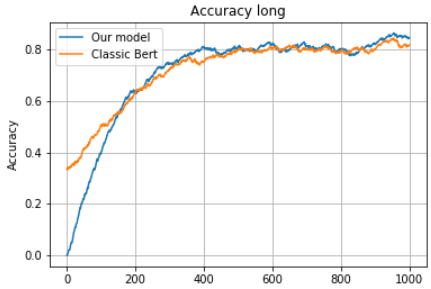
Figure 6: Training performances of ALBERT (A), BERT base with learning rate $1e-5$ and BERT with learning rate $1e-4$.

It would have been interesting to run the same experiments using BERT Large and ALBERT Large, but the memory (RAM) sizes of all the three hardware alternatives

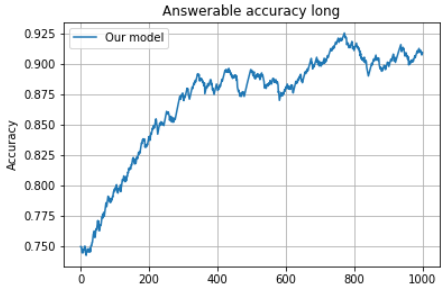


(a) Start short

(b) end short



(c) Start long.



(d) Answerable accuracy.

Figure 7: Partial accuracy of BERT for question answering and our model with the masked loss.

available to us did not allow such experimentation [8].

4.6 Results

This section describes how the experimental phase of **Sapientino** was carried out. We tried many different parameters' configurations and slightly different implementation choices for achieving better results or to reach reasonable trade-offs in the fine-tuning phase.

The measure that was used to evaluate the performances of any model in the Kaggle competition was F1-score, formalized as

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

[8] An attentive reader may notice that the size of ALBERT Large (18M of parameters) is still smaller than the size of BERT base (108M of parameters), so it should be possible to load it into the memory of the devices that allow the load of BERT, but the theory in computer science often differ from the practice and we cannot motivate further.

where *precision* and *recall* are computed as displayed in Figure 8[9].

		Predicted: Positive	Predicted: Negative	
Actual	Positive	True Positive (TP)	False Negative (FN)	**Sensitivity/Recall** $\frac{TP}{TP + FN}$
	Negative	Flase Positive (FP)	True Negative (TN)	**Specifity** $\frac{TN}{TN + FP}$
		Precision $\frac{TP}{TP + FP}$	**Negative Predictive Value** $\frac{TN}{TN + FN}$	**Accuracy** $\frac{TP + TN}{TP + TN + FP + FN}$

Figure 8: Confusion matrix

To be more precise, the submissions to the Kaggle competition are evaluated using micro F1[10] between the predicted and expected answers on a public and private test set. Predicted long and short answers are labeled as "correct" if the token start and end indices of each pattern match *exactly* with the target value, provided that there are multiple correct solutions and the distance is computed with respect tot eh answer that is closes to the predicted one.
For what concerns "yes-no" answers, they are considered as a binary classification task and the computation of precision and recall follows intuitively.

Before discussing our experimental results, it is crucial to explain the different techniques we used in **Sapientino** for selecting start and end position of both short and long answers[11]:

- **Long answer**. Our predictive model is trained to find the start index of the long answers only. We used two different techniques to predict the end of such answers, that follow:
  1. the *optimistic baseline*: that uses the value of end from the target variable that is nearest (in terms of distance from start position) to the prediction. This solution, even if it sounds like "cheating", provides to the authors a reference point, that cannot be outperformed.
  2. the *reasonable solution*: we needed to provide a more general method that can find the best boundaries for a long answer without any further information

[9] Notice that, in the case of short and long answers, the *true positives* are those examples where the predicted indices match one of the possible ground truth indices, the *false positive* are those where the predicted indices do not match one of the possible ground truth indices, or a prediction has been made where no ground truth exists and the *false negatives* are the patterns where no prediction has been made where a ground truth exists.

[10] "micro" F1 takes into account both long and short answers for the computation of precision and recall. In contrast, in "macro" F1 a separate F1 score is computed for each type (short vs long) and then averaged.

[11] Once the answer has been predicted, the start and end tokens are compared with the `[CLS]` tag. If they coincide the answer is discarded.

(in the context of our application, the right answer could not be taken from the target values). The solution we proposed follows this reasoning: if the start token is one of the admissible HTML opening tags (see Section 4.1) then the end index is the position of the corresponding closing tag. If the first token is not an opening tag, a backwards scanning of the crops identifies the closest opening tag that is likely to contain all significant information in the long answer.

In Table 4 the comparison of the two approaches for the long end token.

◇ **Short answer**. Provided that **Sapientino** marks as answerable only a small percentage of short answers, we introduced three techniques to allow some answers (satisfying some properties) to be returned although their score was not high enough. Unfortunately, as shown in Table 2 and Table 3 our heuristics increase the number of question answerable questions, but the accuracy of such answers decreases.

`restoring`: if a short answer was discarded, because it did not reach a fixed threshold in accuracy, it is introduced back in the pool of possible answers *if* it is contained in the *text* of the corresponding long answer;

`matching`: in the pool of possible candidates for short answers there are *only* those answers that are contained in the corresponding long answer;

`mixed`: a combined approach of the first two.

Model1	Public accuracy	Private accuracy	Public N_{long}	Public N_{short}
`mixed`	0.58458	0.58445	274/346	259/346
`matching`	0.59549	0.59610	274/346	146/346
`restoring`	0.58507	0.58839	274/346	263/346
`default`	0.59584	0.60082	274/346	175/346

Table 2: Results of our baseline model on Kaggle's competition with approach 1 for long end token.

Model2	Public accuracy	Private accuracy	Public N_{long}	Public N_{short}
`mixed`	0.53439	0.55292	295/346	254/346
`matching`	0.53439	0.55292	295/346	249/346
`restoring`	0.54699	0.55542	295/346	275/346
`default`	0.54699	0.55542	295/346	275/346

Table 3: Results of our Answerable model on Kaggle's competition with approach 1 for long end token.

5 Conclusions and future work

We describe **Sapientino** as an interactive question answering platform that sees its application in various contexts, such as domotics or education. We performed some experimentation and applied validation techniques to assess the goodness of this

Approach	Public accuracy	Private accuracy	Public N_{start}	Public N_{end}
`Optimistic`	0.59584	0.60082	274/274	274/274
`Reasonable`	0.56799	0.57257	274/274	252/274

Table 4: Results of our best model on Kaggle's competition with all approaches for long end token. The numbers in third and fourth column show the number of guessed token over the upper bound of the correct answer.

model, although we highlight some issues that could be solved more effectively in the future, such as the problem of the presence of a typo in the query (both human error in typing or machine speech-to-text misunderstanding)[12].

Moreover, we would like to stress the fact that choosing to force the model to learn *only* the answerable questions is intended as a simple solution, but we highlight that using a binary classification layer to check if the answer is "plausible" (i.e. the meanings in the question are covered in the answer), as done by [11] and [12] is another technique worth deepening in the future.

Nonetheless, we believe that our approach could be furthermore improved. In fact the good results we obtained with the partial accuracy in Figure 7 are telling us that this approach could be explored in the future. We do not achieve results as good as expected probably due to a sub-optimal policy for selecting the answers and because we do not differentiate between long and short answers. A future work will be introducing a five-categories `[CLS]` classifier, namely `{short, long, yes, no, no answer}` as in [8].

In the future, we intend to use some additional techniques to improve the performances of the model. As done by [10], state of the art model for this task, researchers usually rely on ensembling and bigger model (BERT large is a common choice). However, we did not use these options due to hardware constraints.

Another common technique is to use BERT fine-tuned on SQuAD 2.0 as pre-trained model and then fine-tune it on NQ. However, we did not manage to do follow this path due to lack of availability of BERT base trained on SQuAD with tensorflow.

We would like to stress that, other techniques such as data augmentation did not improve the performances of the model, since the NQ dataset is sufficiently large so as to not require additional examples.

[12] One could try to use library already implemented in Python for auto-correction, such as `auto-correct` available on Pypy, or a more involved and more accurate dictionary-based auto-correct strategy.

References

[1] Konstantin Lopyrev Percy Liang Pranav Rajpurkar, Jian Zhang. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*, 2016.

[2] Percy Liang Pranav Rajpurkar, Robin Jia. *Know What You Don't Know: Unanswerable Questions for SQuAD*, 2018.

[3] Google. *OpenDomain Question Answering Natural questions*, 2019.

[4] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

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

[6] 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. 07 2019.

[7] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

[8] Chris Alberti, Kenton Lee, and Michael Collins. A bert baseline for the natural questions. 01 2019.

[9] See. *submit_full by DeepThought from TensorFlow 2.0 Question Answering competition*, 2019.

[10] Lin Pan, Rishav Chakravarti, Anthony Ferritto, Michael Glass, Alfio Gliozzo, Salim Roukos, Radu Florian, and Avirup Sil. Frustratingly easy natural question answering, 09 2019.

[11] Minghao Hu, Furu Wei, Yu xing Peng, Zhen Xian Huang, Nan Yang, and Ming Zhou. Read + verify: Machine reading comprehension with unanswerable questions. In *AAAI*, 2019.

[12] Seohyun Back, Sai Chetan Chinthakindi, Akhil Kedia, Haejun Lee, and Jaegul Choo. Neurquri: Neural question requirement inspector for answerability prediction in machine reading comprehension. In *International Conference on Learning Representations*, 2020.