Document-grounded Dialogue and Conversational Question Answering

Mauro Comi and Phillip Sloan

January 2022

1 Introduction

In natural language processing (NLP) research, the task of processing a document to be able to respond to queries has become increasingly relevant [4]. The task of a dialogue system is to create a relevant and fluent responses to a previous users turn, this response might also require the conversations previous context. A systems responses also need to provide enough information to answer the query satisfactorily. Figure 1 helps demonstrate this process.

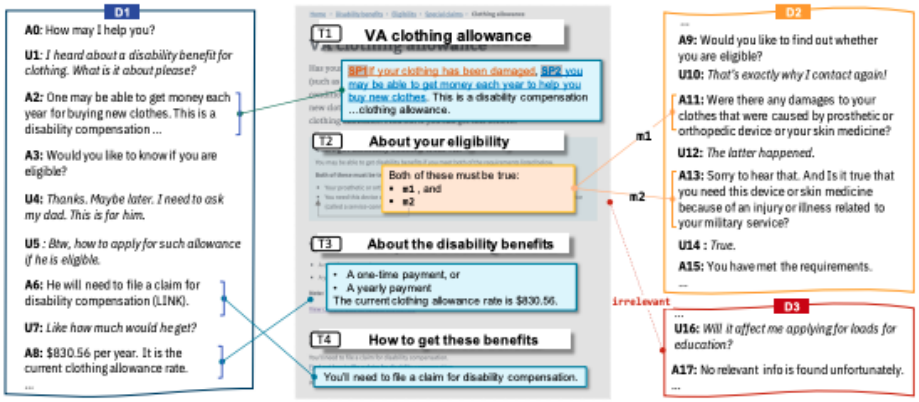

Figure 1: Three sample conversations between an agent and an end user using a website as a grounding document [4]

The first DialDoc Workshop [3] focuses on grounding and creating agent responses in such natural language processing systems, and has created two shared tasks. The first task is knowledge identification, from a given document and a user utterance and context, the relevant information is gleaned from the document and the relevent response span(s) are generated. The second task is response generation, which uses the spans from the previous task to generate a coherent response to the users turn.

The aim of this report is to tackle the first task. To give more detail. First we describe our data exploration procedure, and what we understood from this. Second, we describe our model, going how we selected our model to fine-tuning and the techniques we developed. Finally we provide the results of our model and discuss the results and possible improvements.

2 Dataset and Data Exploration

The doc2dial dataset was created by Feng et al [4]. This dataset contains 4,793 annotated dialogues based on a total of 487 documents. The dialogues are conversations between agents and an end user which are grounded within the associated document. These documents were acquired from various American government service websites, including Social Security Administration (ssa), Department of Motor Vehicles (dmv), United States Department of Veterans Affairs (va), and Federal Student Aid (fsa) [4]. Each document in the dataset is associated with one of these domains. Each document has several dialogues between a user and an agent. The documents within the dataset are generally quite large, with a median length of 817 words which can be split into 991 tokens [6].

During data discovery we looked in-depth into this dataset, following dialogues and locating spans within the grounding document to allow us to understand their context within the document. The first thing we noticed was that with the user utterances, there were sometimes some spelling mistakes which will increase the difficulty of the task. An example of a spelling mistake is shown in the next example. Moreover, we realised there could be a potential improvements in the span selection. Some spans potentially miss some part of a sentence which could potentially be useful for a response. For example, when for the user utterance:

> 'Hello, I forgot o update my address, can you help me with that?'

Which the agent responds with spans 6 and 7 from their relevant document, these spans are:

> 'you must report a change of address to DMV within ten days of moving. That is the case for the address associated with your license, as well as all the addresses associated with each registered vehicle, which may differ.'

However when investigating the relevant document, we noticed that some of the first sentence was missing, this was span 5. Adding this to the other two spans (span 5 in red):

> 'By statute , you must report a change of address to DMV within ten days of moving. That is the case for the address associated with your license, as well as all the addresses associated with each registered vehicle, which may differ. '

When discussing this, we felt that span 5 would have been important to the response to the user. For example, in the United Kingdom there is the Highway Code which is not considered law which often confuses people about whether they actually have to follow the instruction or not[5]. A user might think this advice is advisory in a similar vein, by stating that this is statute or law would allows the user to know this is a legal requirement.

On further investigation we came to the conclusion that they seem to be creating spans by splitting on various punctuation which often split up sentences into smaller chunks. When discussing this we felt that including entire sentences could provide users with more information, and provide more

insightful answers. This decision led us to using the doc2dial *dialogue_domain* and *document_domain* to create our own dataset, rather than using the *doc2dial_rc* dataset. The creation of our dataset instead of using doc2dial_rc increased the difficulty in making a training and validation set, but we felt it was justified by having the full sentences as spans instead. How we implemented our dataset can be seen in Listing B in Appendix B.

3 Model

3.1 Model Selection

The goal of this task is to return a span of the grounding document based on the user queries. Mathematically, this corresponds to generating two probability distributions over the tokens, representing the probability of each token to be the initial and final element of this span. Simple classifiers, such as the Naive Bayes classifier, are able to compute these probability distributions. However, they are not able to generalise on unseen tokens, and this limits their applicability to relatively simple use cases. For this reason, Neural Network models are preferred: they can handle very large input-output size mappings and generalise over unseen data. In this report we use DistilBERT [2], a faster and lighter version of BERT [1]. Like BERT, DistilBERT is a stack of transformer encoders. It receives as input a vector of tokens in the form:

$$[CLS] + \text{Question} + [SEP] + \text{Context} + [SEP]$$

and an attention mask where the tokens related to the questions are mapped to 1s and those related to the context are mapped to 0s. These vectors are sent to three initial embedding layers constructed from the Token Embeddings, Segment Embeddings, and Position Embeddings. The output of these embeddings is processed by ten encoder layers, each consisting of an attention layer, a layer normalisation module and a fully connected network. The output of these ten encoder layers is mapped into two different probability distributions, one for the start tokens and one for the end tokens. This is done by a fully connected network that is specifically trained for Q&A tasks.

The BERT model used for the baseline by [4] truncates any document that is greater than 1024 tokens, which will potentially cause some of the required spans for task one to be removed. While we were happy with using a BERT model, losing some of the document was not considered acceptable as this would mean we would not have all the available spans. Several researches got around this limitation of the models by using sliding windows [6], which is something we felt would be interesting to implement. The implementation of this can be found in Section 3.3.1.

3.2 Fine-tuning

Our baseline is the F1 score from using the pretrained DistilBERT model from HuggingFace [2]. This model was previously trained on the BookCorpus dataset [8] and on English Wikipedia for 90 hours. This training phase allowed the model to acquire a solid understanding of the English language. For this reason we fine-tuned the network by freezing every layer but the final fully connected network, which is built on top of the encoder layers. We froze these layers as we did not want to affect the network parameters optimised during the pretraining phase. Also, training the full model would have been computationally very expensive and reusing the pretrained parameters significantly reduces the

training time to a more manageable level. The hyper-parameters we selected to train the algorithm on are shown in Table 1:

Parameter	Learning Rate	N. Epochs	Batch Size	Weight Decay
Value	$3e-4$	5	16	0.1

Table 1: Hyper-Parameters used for the DistilBERT Model

We trained our model on a single GPU, an NVIDIA GTX 1050 for 6 hours, with an unexpected windows reboot in the middle. We attempted to train our model on Google Colab however the limitations of the standard version meant that we kept hitting the maximum limit of GPU usage, which meant we could not use it again for an undefined amount of time.

Tab. 2 shows the intermediate training and validation loss over the first 4 epochs. The validation set was obtained by extracting a random split consisting of 33% of the training data. Both the training and validation loss are decreasing over time and there is no sign of convergence. This means that a longer training time would potentially improve the performance of our model. However, this was not possible due to the size of the model and the lack of required computational power, as we have mentioned in this section.

Epoch	Training Loss Rate	Validation Loss
1	6.118200	6.095091
2	6.101500	6.082878
3	6.083700	6.065759
4	6.067500	6.057233

Table 2: Training loss and validation loss during the fine-tuning of the DistilBERT model

3.3 Techniques

3.3.1 Sliding Windows

One of the limitations of DistilBERT for this Q&A task, is that it only allows 512 tokens to be input into the model, which is half the amount that BERT allows. When researching this issue, we noticed that other researchers tackled this with an approach called sliding windows with stride [6]. With this approach the document is split into multiple windows, enabling the entire document to be captured by having the model run multiple times on each window. We felt it was important to include the entire document, to ensure we do not miss any potential predictions, because of this we thought it would be important to implement our own version of a sliding window algorithm. Algorithm 1 demonstrates the implemented algorithm.

This algorithm takes in the question, which is the last utterance and the context of any previously unanswered utterances and the relevant document this conversation is grounded in. The stride is an optional input, which defaults to 256 tokens. If *start_token* and/or *end_token* are not given, then the algorithm will split a document into several different windows and return all of these windows. If a *start_token* and an *end_token* are provided then only the sliding window(s) that cover both of these tokens will be returned.

The if statement on line four handles potential edge cases when a document is smaller than DistilBERTs input of 512 tokens. The variable *training* on line seven is a boolean variable, which evaluates to true only when a *start_token* and an *end_token* have been given. The variable question_inside_context is another boolean variable, checking if the start and end token are within this window. Line 18 is a function that adds a '[SEP]' token to the end of a window, if it not already there, ensuring each window satisfies the input requirements of DistilBERT for a Q&A task.

Algorithm 1: Sliding Windows Algorithm

Input: question, document, stride=256, start_token=None, end_token=None
Output: windows: a list of documents with a length of 512 tokens.

```
1  windows = list()
2  start=0
3  max_token_length=511
4  end=max_token_length − length(question) if length(document) ≤ max_token_length −
     length(question) then
5  |  end = length(document)
6  while start ≥ length(document) do
7  |  training= start_token ! = None and end_token ! = None if training then
8  |  |  question_inside_context = start_token ≥ start and end_token ¡= end
9  |  if not training or training and question_inside_context then
10 |  |  window = document[start:end]
11 |  |  windows.append(window)
12 |  if end == length(document) then
13 |  |  break
14 |  start = start + stride if length(document) − start  max_token_length − len(question)
   |   then
15 |  |  end = length(document)
16 |  else
17 |  |  end = end + stride
18 windows = add_sep_tokens(windows)
19 return windows
```

3.3.2 Mask

Even though our model predicts probabilities over all tokens in each window, as previously mentioned in Chapter 1, we wanted to limit this so that our model only returns complete sentences. We made this possible by creating a mask, selecting for the first word of a sentence or the last word. Our implementation of this is demonstrated by Algorithm 2.

The algorithm uses PyTorch, an open source machine learning framework [7]. Line one of the algorithm creates a mask using the tensor *input_ids*. The mask sets everything that has the token id of a full stop ('.') to 1, and everything else to zero. Line two uses the tensor *segment_ids* to mask out the question, as we do not want this to be a potential answer to the question. Finally,

the if statement on line three rolls the mask to the right if the mode is 'start' otherwise it rolls the mask to the left. By doing so, only the tokens after a full stop (corresponding to the first word of each sentence) are not masked out when the mode is 'start', while only the tokens before a full stop (corresponding to the last word of each sentence) are not masked out when the mode is 'end'. *input_ids* is the tokenized input, this is 512 tokens. This tensor contains all the tokens related to the current user query and the context returned by the sliding windows algorithm. *segment_ids* is a type of mask that allows the model to differentiate between the question and the answer within the *input_ids*. *mode* is a string, either 'start' or 'end' to denote if we are looking for the start or end of a sentence respectively.

Algorithm 2: Mask Algorithm

Input: input_ids, segment_ids, mode
Output: mask: a mask for the start and end logits.

```
1 input = torch.where(input_ids == 1012, 1, 0)
2 input = input * torch.tensor(segment_ids)
3 if mode == 'start' then
4 |   mask = torch.cat((torch.tensor([0]),a),0)[:-1]
5 else
6 |   mask = torch.cat((a, torch.tensor([0])),0)[1:]
7 return mask
```

4 Results and Discussion

In this section we describe the results of our models predictions. For our quantitative results we use the scripts provided by the doc2dial workshop [3], this script provides the token-level F1 score which is an indication of how well our predicted span matches against the reference span. We also perform a qualitative review of our predicted span outputs to try and understand how we could improve on our predictions.

4.1 Quantitative

Table 3 gives a summary of our main results for the doc2dial shared task 1, using the doc2dial scripts provided. The results fall short (by some margin!) of the baseline provided by the doc2dial baseline [4]. We feel there are several factors that are causing the results of our model to under perform on the provided scripts. Firstly, we used DistilBERT instead of BERT, we did this because it is a slimmed down version and therefore quicker to run. This could cause some loss of accuracy in our predictions, so could be a factor, but is obviously not enough for the F1 score to reduce this much.

The second reason we theorised was that our predictions are entire sentences, which the spans within the doc2dial dataset are not. As we were aiming to provide more information, even if our model gets the prediction right it would provide more data than the script deems necessary, providing a negative impact on the score. Another issue with our decision to only predict full sentences was that as the doc2dial dataset we trained our model on did not use full sentences as its answers.

Model	F1 score
DistilBERT before fine-tuning, no mask	7.73
DistilBERT before fine-tuning, mask	10.98
DistilBERT after fine-tuning, no mask	14.71
DistilBERT after fine-tuning, mask	13.19

Table 3: Results of our model, before and after fine-tuning, with and without a mask

Interpreting the results, we can see that our mask algorithm seems to improve the F1 score before fine-tuning but negatively impacts the result after fine-tuning. A possible improvement would be to include this technique in the training procedure, as at the moment it is applied as a post processing technique. Training and fine-tuning the model has a positive impact, almost doubling the F1 score when no mask is applied.

4.2 Qualitative

In this section we qualitatively evaluate the predictions from our model, to try and gain insights into what went right and what went wrong, looking for clues for how we could improve the model. We provide several examples to help describe what we found, each example comes with our prediction and reference span from the doc2dial validation set.

4.2.1 Example One

Prediction: '.? help with the most common questions about the fafsa process'
Reference: 'Help with the most common questions about the FAFSA process. '

In this first example we notice that our prediction is incredibly close to the reference span, however, the prediction has some unnecessary punctuation at the start, and the 'FAFSA' acronym is lowercase rather than uppercase. Having checked this prediction alone, despite being slightly different, this span received an F1 score of 100 from the doc2dial script. This suggests that punctuation and the case of the text is ignored by the script.

4.2.2 Example Two

Prediction: 'what kind of insurance do i need? new york law requires that you have auto liability insurance coverage. the minimum amount of liability coverage is $ 10, 000 for property damage for a single accident $ 25, 000 for bodily injury and $ 50, 000 for death for a person involved in an accident $ 50, 000 for bodily injury and $ 100, 000 for death for two or more people in an accident your liability insurance coverage must while the registration is valid, even if you don t use the vehicle there are exceptions for motorcycles 1 remain in effect be new york state insurance coverage, issued by a company licensed by the ny state department of financial services and certified by ny state dmv - out - of - state insurance is never acceptable be issued in the name of the vehicle registrant, and remain in the name of the registrant at all times 2 the new york state

department of financial services website has more information about liability insurance and insurance companies licensed in new york : http : / / www. dfs. ny. gov [1]'
Reference: 'We will never accept out - of - state vehicle insurance coverage of any type. '

In this second example, even at a glance it is noticeable that our prediction misses the mark and is significantly inaccurate, the F1 score is recorded as 9 which reflects the poor result. For this dialogue, the user utterance was: 'i'm enquiring about out of state insurance docs'. When reviewing the predicted span, it does mention out of state insurance, but semantically does not help the user. Perhaps if we had been able to train the model for longer, or been able to fine-tune the hyper-parameters then we would have been able to predict with more accuracy.

4.2.3 Example Three

{'id': '9a45905f9c4bb1f095e1039dc0387be0_12',
'prediction_text': '1 learn about enhanced, real id and standard documents [1]'},
{'id': '9a45905f9c4bb1f095e1039dc0387be0_14',
'prediction_text': '1 learn about enhanced, real id and standard documents [1]'},
{'id': '07e9d1fe776f3431e82683047ee21968_1',
'prediction_text': '1 learn about enhanced, real id and standard documents [1]'},
{'id': '07e9d1fe776f3431e82683047ee21968_3',
'prediction_text': '1 learn about enhanced, real id and standard documents [1]'}

When broadly reviewing the predictions, we noticed that a lot of user utterances received the same answer. We looked into the issue and noticed all of these 'id's' were consecutive and were related to the same document. For this reason we think that this might arise because we did not shuffle the training data when fine-tuning the model. By not shuffling the data many consecutive instances are very similar, with the only thing changing being the user utterance. Therefore we are introducing a bias into the training because the inputs to the model are so similar.

4.3 Conclusion

The lack of computational power was a problem for us, Daheim et al [6] noted that they were unable to run the doc2dial baseline on their own systems, complaining of memory constraints. For reference, they had multiple GPUs including a NVIDIA GTX 1080 Ti and an RTX 2080 Ti's. Considering the only systems we had available were a system with an NVIDIA GTX 1050 and Google Colab, which often would shut us out, we had to pick an option that was not as greedy computationally. For this reason we did not get a chance to hyper-parameterise our model fully, which would have improved its performance, providing more accurate results.

The implementation of our masking technique resulted in a better performance for our baseline model, but not for our fine-tuned model. This suggests that masking out those tokens that are not relevant for the QA task might be a useful technique which requires improvement. A possible action in this regard would be to embed the masking procedure in the training process, instead of only using it as a post-process approach. However, the DistilBERT model we are using was pretrained

without this technique, so we don't know how embedding this technique during fine-tuning would affect its output predictions.

During this assignment, we spent a significant amount of time understanding and building our own dataset. While we felt that this would provide us with answers that would potentially help the end users more, it did impact the amount of time that we had to train and improve our model. If we had used the doc2dial_rc dataset, we would have been able to spend more time training and fine-tuning. This could have improved our performance, as more training time was found to be the largest factor in improving the F1 score.

We used the BertDistilledTokenizerFast to tokenize our data, and used the HuggingFace manual to understand how to use this tokenizer. On looking further into the documentation it has been noticed that these tokenizers might already have the sliding windows functionality, from their parent tokenizer class [0]. This again would have saved a lot of time, and given us more time to be able to train and fine-tune the model for a better accuracy than the baseline.

Despite both of these issues causing a time-sink, where there were possible quicker solutions, we feel we gained a lot from doing both of these steps. As we gained a deeper understanding of the data, and how certain algorithms are implemented behind the scenes.

References

[1] Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. NAACL HLT.

[2] Sanh, Victor, et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." arXiv preprint arXiv:1910.01108 (2019).

[3] Song Feng. 2021. Document-grounded Dialogue and Conversational Question Answering. https://doc2dial.github.io/workshop2021/shared.html.

[4] Feng, Song and Wan, Hui and Gunasekara, Chulaka and Patel, Siva and Joshi, Sachindra and Lastras. Luis, 2020. doc2dial: A Goal-Oriented Document-Grounded Dialogue Dataset in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 8118-8128. Association for Computational Linguistics.

[5] RAC. 2018. 16 Highway Code rules you shouldn't ignore. https://www.rac.co.uk/drive/advice/legal/16-highway-code-rules-most-people-ignore.

[6] Daheim, Nico and Thulke, David and Dugast, Christian and Ney, Hermann. 2021. Cascaded Span Extraction and Response Generation for Document-Grounded Dialog in Proceedings of the 1st Workshop on Document-grounded Dialogue and Conversational Question Answering (DialDoc 2021). pp. 57-62. Association for Computational Linguistics.

[7] Paszke, A. et al., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035. Available at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[8] Zhu, Yukun, et al. "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books." Proceedings of the IEEE international conference on computer vision. 2015.

HuggingFace, 2020. https://huggingface.co/transformers/v2.11.0/main$_c$*lasses/tokenizer.html*

A GitHub Repository link

The code for our full implementation can be found at our GitHub here.

B Implementation of our Dataset

```
tokenizer = DistilBertTokenizerFast.from_pretrained('bert-base-uncased')

# Defining train_dict
train_dict = dict()
train_dict['train_document'] = []
train_dict['train_id_sp'] = []
```

```
train_dict['train_user_utterance'] = []
train_dict['train_doc_domain'] = []
train_dict['train_doc_id'] = []
train_dict['train_text_sp'] = []
train_dict['train_dial_id_turn_id'] = []      # necessary for evaluation
train_dict['train_start_pos'] = []
train_dict['train_end_pos'] = []
train_dict['train_start_tok'] = []
train_dict['train_end_tok'] = []
train_dict['train_all_utterances'] = []

start = time.time()
for idx, dialogue in tqdm(enumerate(dialogue_dataset)):
    #if idx == 100:
    #    break
    dial_id_turn_id = []        # running list of <dial_id>_<turn_id> for evaluation
    sp_id_list = []             # running list of spans per document
    user_utterance_list = []    # running list of user utterances per document
    all_utterances_list = []
    for idx_turn, turn in enumerate(dialogue['turns']):
        dial_id_turn_id.append(dialogue['dial_id'] + '_' + str(turn['turn_id']))
        all_utterances_list.append(turn['utterance'])
        if turn['role'] == 'user':
            # If the previous turn was still the user, we want to concatenate the
            # current and previous utterances
            if (idx_turn > 0 and dialogue['turns'][idx_turn-1]['role'] == 'user'):
                turn['utterance'] = tokenizer(turn['utterance'], padding=True, truncation=True, retur
                previous_utterance = user_utterance_list[-1][:-1]  # the previous utterance ends with
                turn['utterance'] = torch.cat((previous_utterance, turn['utterance']), 0)
                user_utterance_list[-1]= turn['utterance']  # replace last element in list
            # If the last utterance is by the user, we don't store it
            elif idx_turn == len(dialogue['turns'])-1:
                continue
            else:
              # TURN UTTERANCE IS FLATTENED AND ONLY THE [INPUT_IDS] IS STORED
              turn['utterance'] = tokenizer(turn['utterance'], padding=True, truncation=True, return_
              user_utterance_list.append(turn['utterance'])   # adding user utterance to user_utterar
        else:
            references = turn['references']
            ref_sp_id = []
            for ref in references:
                ref_sp_id.append(ref['sp_id'])
            sp_id_list.append(ref_sp_id)   # adding list of sp_ids per dialogue to list of sp_ids per

    train_dict['train_id_sp'].append(sp_id_list)
    train_dict['train_user_utterance'].append(user_utterance_list)
    train_dict['train_all_utterances'].append(all_utterances_list)
    train_dict['train_doc_domain'].append(dialogue['domain'])
    train_dict['train_doc_id'].append(dialogue['doc_id'])
    train_dict['train_dial_id_turn_id'].append(dial_id_turn_id)

    for doc in document_dataset:
        if doc['doc_id'] == train_dict['train_doc_id'][-1]:
            doc['doc_text'] = tokenizer(doc['doc_text'], padding=True, truncation=False, return_tenso
            train_dict['train_document'].append(doc['doc_text'])          # adding the total document
```

```
            text_sp_2 = []
            start_sp_list = []          # big start sp list
            end_sp_list = []            # big end sp list
            start_tok_list = []          # big start token list
            end_tok_list = []            # big end token list
            for train_spans_id in train_dict['train_id_sp'][-1]:
                text_sp = ""
                ref_start_pos_list = []
                ref_end_pos_list = []
                for span in doc['spans']:
                    if span['id_sp'] in train_spans_id:
                        text_sp += span['text_sp']
                        ref_start_pos_list.append(span['start_sp'])
                        ref_end_pos_list.append(span['end_sp'])
                start_pos = np.amin(ref_start_pos_list)
                start_sp_list.append(start_pos)
                # convert start_pos to start_token
                start_tok_pos = doc['doc_text'].char_to_token(start_pos)
                # check that start_tok_pos is not None, if it is go to the next character
                while start_tok_pos == None:
                    start_pos = start_pos + 1
                    start_tok_pos = doc['doc_text'].char_to_token(start_pos)
                start_tok_list.append(start_tok_pos)
                # convert end_pos to end_token
                end_pos = np.amax(ref_end_pos_list)
                end_sp_list.append(end_pos)
                end_tok_pos = doc['doc_text'].char_to_token(end_pos)
                # check that end_tok_pos is not None, if it is go to the next character
                while end_tok_pos == None:
                    end_pos = end_pos - 1
                    end_tok_pos = doc['doc_text'].char_to_token(end_pos)
                end_tok_list.append(end_tok_pos)
                text_sp_2.append(text_sp)
            train_dict['train_text_sp'].append(text_sp_2)
            train_dict['train_start_pos'].append(start_sp_list)
            train_dict['train_end_pos'].append(end_sp_list)
            train_dict['train_start_tok'].append(start_tok_list)
            train_dict['train_end_tok'].append(end_tok_list)
            break
end = time.time()
print(f'Total time: {end-start}')
```