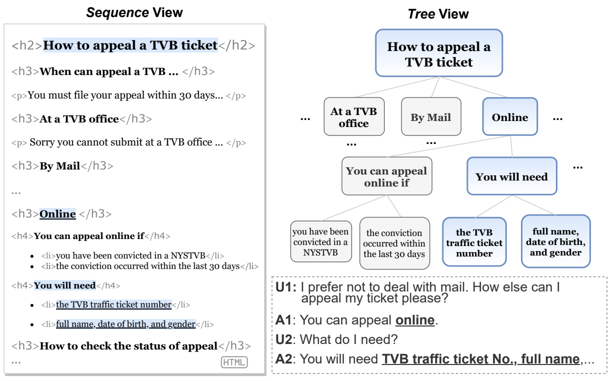
Does Structure Matter? Encoding Documents for Machine Reading Comprehension

Hui Wan Song Feng Chulaka Gunasekara Siva Sankalp Patel
Sachindra Joshi Luis A. Lastras
IBM Research AI
{hwan@us, sfeng@us, chulaka.gunasekara@}.ibm.com
{siva.sankalp.patel@, jsachind@in, lastrasl@us}.ibm.com

Abstract

Machine reading comprehension is a challenging task especially for querying documents with deep and interconnected contexts. Transformer-based methods have shown advanced performances on this task; however, most of them still treat documents as a flat sequence of tokens. This work proposes a new Transformer-based method that reads a document as tree slices. It contains two modules for identifying more relevant text passage and the best answer span respectively, which are not only jointly trained but also jointly consulted at inference time. Our evaluation results show that our proposed method outperforms several competitive baseline approaches on two datasets from varied domains.


Sequence View
Tree View
How to appeal a TVB ticket
At a TVB office
By Mail
Online
You can appeal online if
You will need
you have been convicted in a NYSTVB
the conviction occurred within the last 30 days
the TVB traffic ticket number
full name, date of birth, and gender
HTML

```
<h2>How to appeal a TVB ticket</h2>
<h3>When can appeal a TVB ... </h3>
<p>You must file your appeal within 30 days... </p>
<h3>At a TVB office</h3>
<p> Sorry you cannot submit at a TVB office ... </p>
<h3>By Mail</h3>
...
<h3>Online </h3>
<h4>You can appeal online if</h4>
  • <li>you have been convicted in a NYSTVB</li>
  • <li>the conviction occurred within the last 30 days</li>
<h4>You will need</h4>
  • <li>the TVB traffic ticket number</li>
  • <li>full name, date of birth, and gender</li>
<h3>How to check the status of appeal</h3>
...
```

U1: I prefer not to deal with mail. How else can I appeal my ticket please?
A1: You can appeal **online**.
U2: What do I need?
A2: You will need **TVB traffic ticket No., full name**,...

Figure 1: A sample document segment with the hierarchical structure (left), the partial tree slices (right) and sample dialogue turns (bottom right).

1 Introduction

Machine Reading Comprehension (MRC) is the task of reading a given text and answering questions about it (Liu et al., 2019). Some MRC tasks such as SQuAD (Rajpurkar et al., 2016, 2018),and ShARC (Saeidi et al., 2018) provide a short text snippets as the context documents; while others such as TriviaQA (Joshi et al., 2017), Natural Questions (Kwiatkowski et al., 2019) and Doc2Dial (Feng et al., 2020) use full articles as documents. Most top performing models on MRC tasks use different variants of Transformers (Vaswani et al., 2017). Transformer-based models typically only consider a certain number of tokens, utilize a sliding window approach (Richardson et al., 2013) or segment the document into passages (Hu et al., 2019; Wang et al., 2019) due to the constraint on the size of input sequence. More recent works explore how to scale up input length (Yang et al., 2019; Beltagy et al., 2020; Kitaev et al., 2020; Wang et al., 2020; Ainslie et al., 2020) but still mainly focus on flat sequences. In addition to scaling up input length, ETC(Ainslie et al., 2020) also propose to deal with encoding structured inputs based on relative position encoding (Shaw et al., 2018) through the global-local mechanism.

A series of recent work explores incorporating structured knowledge embedded in text into MRC (Shen et al., 2020; Dhingra et al., 2020). However, such kind of linking information for creating triples is not necessarily prominent in documents other than Wikipedia. Some works segment the document content based on its semantic structures and rank them based on their relevance to the query (Yan et al., 2019; Lee et al., 2018; Wang et al., 2018; Zheng et al., 2020; Liu et al., 2020).

Another thread of works, on hierarchical document encoding (Li et al., 2015; Yang et al., 2016; Zhang et al., 2019; Guo et al., 2019), first obtain sentence level representations then encode document based on the sentence vectors. Those works do not directly apply on fine-grained answer extraction across sentences.

In many online documents, certain important information unfolds through the semantic relations of hierarchical structures such as parent-child and siblings between different parts of the document. Figure 1 illustrates the difference when using a doc-

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 4626–4634
June 6–11, 2021. ©2021 Association for Computational Linguistics

ument with and without the structure information for a MRC task. For query `U1`, it is crucial to keep in mind we are in the context of “How to appeal a TVB ticket” and “Online” while reading the passage of “You will need” to find the answer to the user query. However, conventional Transformers fail to capture such contextual information when the text is too long to fit in the maximum sequence length allowed.

In this work, we explore the utilization of document structure for the focused task of fine-grained Machine Reading Comprehension on document. We propose a Transformer-based method that reads a document as tree slices; it jointly learns the relevance of paragraphs and spans, and then performs a cascaded inference to find the best answer span. Our work is intuitively inspired by how people read through documents (Choi et al., 2017) based on structural cues such as titles and subtitles, and then focus on the relevant parts to search for an answer. We utilize the structural information naturally available in online documents for identifying tree slices. Each slice corresponds to nodes along a path from a root node to a lower level child node as illustrated by the right part of Figure 1. Thus, we are able to capture the essential structural information for the inference that could be outside of a conventional sliding window or text segment. Compared to approaches such as Longformer (Beltagy et al., 2020) or ETC (Ainslie et al., 2020), our approach can be directly applied to many existing pretrained models, and has a small GPU memory footprint. RikiNet (Liu et al., 2020) employs a dynamic paragraph dual-attention reader and a multi-level cascaded answer predictor, while our tree slices consider hierarchical structures above paragraphs, and our cascaded inference is in beam search style rather than greedy decoding style in RikiNet.

We evaluate on two datasets with structured documents: one obtained from Natural Questions (Kwiatkowski et al., 2019), which is based on Wikipedia articles, and one from Doc2Dial (Feng et al., 2020), which is based on web pages of several domains. Our proposed method is compared with several baselines to see performance gain on both datasets. For example, our method achieves 4% gain of F1 on Doc2Dial, which shows its superiority on small-scaled dataset across multiple domains.

Our contributions can be summarized as follows: (1) We propose a Transformer-based method that reads a document as a tree. It simultaneously identifies the relevance of paragraphs and finds the answer span via jointly trained models with cascaded inference. (2) Our method can utilize common structures as seen in many web documents. It allows Transformer models to read in more focused content but with deep context; thus it can be used to handle long documents in an efficient way. (3) Our proposed method outperforms several competitive baseline methods on two kinds of MRC tasks with documents from varied domains.

2 Approach

We adopt a Transformer-based document-tree-slice encoder with joint learning and cascaded inference. Our approach is influenced by the pattern of human behavior during reading (Choi et al., 2017), which is to focus on a smaller portion at a time and favor the more relevant parts while looking for answer. This approach can also overcome the constraint on fixed-length input allowed by the common Transformer architecture (Vaswani et al., 2017). More importantly, this enables us to always include important structural context information during encoding.

2.1 Tree Slicing

To obtain the tree representation of a web page, we consider the different levels of HTML title tags as the main indicators of the hierarchical structures such as parent-child and siblings in Figure 1. More details are provided in Section 3.

Formally, we define an example in the dataset as (Q, D, s, e) where Q is a question, D is a document, s and e denote the inclusive indices pointing to the start and end of the target answer span.

Suppose one does not consider the structure information, D is treated as a sequence and sent to Transformer encoder. For long documents, the sliding window approach is widely used to truncate D into m overlapping fragments $D_1, ...D_m$, and (Q, D, s, e) is converted to m training instances (C_i, s_i, e_i) where $C_i = ([\texttt{CLS}], Q_1, ..., Q_{|Q|}, [\texttt{SEP}], D_{i,1}, ..., D_{i,|D_i|}, [\texttt{SEP}])$, s_i and e_i are mapped indices in C_i. If D_i does not contain the target answer, s_i and e_i are set to the index of the [CLS] token.

In our proposed approach to encode a document, we consider the structured information along with its content. Given a document D, let k be

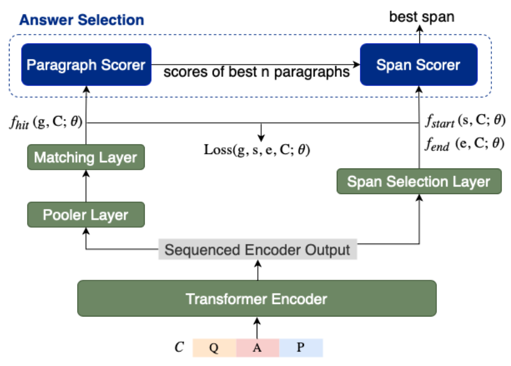


Figure 2: Joint Model with Cascaded Inference.

the number of leaf nodes in its tree structure. We first convert (Q, D, s, e) into k examples (Q, A_i, P_i, s_i, e_i), where P_i is a leaf node, s_i and e_i are mapped indices in P_i, and A_i denotes P_i's ancestor chain in the document tree of D. Each (Q, A_i, P_i) is then encoded with Transformers as a sequence $C_i = ([\texttt{CLS}], Q_1, ..., Q_{|Q|}, [\texttt{SEP}], A_{i,1}, ..., A_{i,|A_i|}, [\texttt{SEP}], P_{i,1}, ..., P_{i,|P_i|}, [\texttt{SEP}])$. An example A_i in Figure 1 would be the list of {*'How to appeal a TVB ticket conviction'*, *'Online'*, *'You will need'*}. Intuitively, the tree slice approach ensures that the most relevant structural information, the ancestor chain, is always taken into account and attended to with Transformer encoder, while this is unlikely to be guaranteed by the sliding window truncation.

2.2 Joint Model with Cascaded Inference

With tree slicing approach, from each document we have many paragraphs to select the answer span from, as compared to the case of sliding windows. In order to teach the model to favor the candidates from the more relevant parts of the document, we train a joint model to simultaneously learn to identify the relevance of paragraphs and find the answer span. Then we perform a cascaded inference to first find the most relevant paragraphs and then find the best answer span from them, based on the scores from the joint model, as Figure 2 shows.

Joint model The encoded representation of C can be used to perform two tasks, each being handled by a separate module: 1) the pooler layer and the matching layer (both linear layers) predict how likely a paragraph P contains the answer; 2) the span selection layer (another linear layer) identifies the answer span from P. Each training instance is converted to (C, s, e, g) where $g \in \{0, 1\}$ denotes whether P contains the answer. We define the loss function to be

$$\begin{aligned} Loss(g, s, e, C; \theta) = &\, \mathcal{L}_{\texttt{CE}}(f_{hit}(g, C; \theta)) \\ &+ \lambda * (\mathcal{L}_{\texttt{CE}}(f_{start}(s, C; \theta)) \\ &+ \mathcal{L}_{\texttt{CE}}(f_{end}(e, C; \theta))) \end{aligned}$$

where $\mathcal{L}_{\texttt{CE}}$ is the Cross Entropy loss function, θ denotes the model parameters, and each f is the score obtained by the corresponding linear layer on top of the last layer representation of Transformer encoder: f_{hit} by the pooler layer and the matching layer, and f_{start} and f_{end} by the span selection layer.

Cascaded inference After the two modules of the model are jointly trained, we conduct a cascaded inference in a beam search style.

- First, from all the instances corresponding to tree slices of a single document, we select the top n instances ranked by $f_{hit}(g = 1, C; \theta)$. This is important for filtering out high scored spans from irrelevant tree slices.

- Then, from these top instances, each candidate document span is assigned a score attributed from both modules of the model:

$$\begin{aligned} \texttt{Score}(C, s, e) = &\, f_{hit}(g = 1, C; \theta) \\ &+ \gamma * (\texttt{Score}_{start}(s, C; \theta) \\ &+ \texttt{Score}_{end}(e, C; \theta)) \end{aligned}$$

  where we adopted the trick from (Alberti et al., 2019) to define $\texttt{Score}_{start}(s, C; \theta) = f_{start}(s, C; \theta) - f_{start}(\texttt{Idx}_{\texttt{CLS}}, C; \theta)$, and $\texttt{Score}_{end}(e, C; \theta)$ as $f_{end}(e, C; \theta) - f_{end}(\texttt{Idx}_{\texttt{CLS}}, C; \theta)$.

- Finally, we choose the document span with the highest $\texttt{Score}(C, s, e)$ as the answer.

Given a document with tree slices, we would create more instances than the sliding window approach. However, with the joint training and cascaded inference, our model reaches better accuracy in less training time, as will be shown in Section 4.

3 Data

Our focused task is utilizing document structure in contextual representation for fine-grained MRC. Since there is very few prior MRC datasets that provides document structure information, we identified two public datasets where HTML markup tags are available in the document data together with QA pairs, and extract tree structure out of the HTML documents for MRC. Data script could be found at http://html2struct.github.io.

Extract Tree Structure To obtain the tree representation of documents from the two datasets, we first parse HTML files to get markup tags of the textual content elements, which corresponds to the titles, lists, tables and paragraphs. We consider the different levels of title tags as the main indicators of the hierarchical structures such as parent-child and siblings. Thus, the stem nodes are inherently section or subsection titles of the article and leaf nodes are typically paragraphs, list content or table content. We assign the article title as the tree root. Please refer to Appendix A for more details about the data statistics for the experiment.

NQStruct Natural Question (Kwiatkowski et al., 2019) provides QA pairs that are grounded in Wikipedia articles. The original task provides answers in two formats: *long answer*, typically a paragraph or table; *short answer*, typically one or more entities. In our task, we focus on identifying the short answer given the whole document as the input, and do not use the long answers data. We observe the bias on answers appearing in first paragraph, which is significant enough to serve as a baseline (Kwiatkowski et al., 2019). Thus, we follow Geva and Berant (2018) to alleviate such bias by only considering the questions where the short answer does not come from the first paragraph. As a result, we derive a subset of 48K examples from about 100K examples with short answers from training and dev sets.

D2DStruct Doc2Dial (Feng et al., 2020) provides document-grounded dialogues with annotations of dialogue scenes, which allow us to identify question-answer pairs that are most related to our target task. Specifically, we combine each turn of the agent responding to a user query, together with the previous dialogue context, as a question. The public dataset contains over 4.1K document-grounded dialogues based on about 450 documents from different domains, and we derive 9.3K QA pairs out of it.

4 Experiments and Results

We compare our proposed method (**TreeJC** in short) with several baseline methods. Next we describe the baselines, the experiment settings, and present the evaluation results.

Model	F1	Exact Match	Train hrs
SW	49.6 ± 0.1	32.2 ± 0.2	1.75
Longformer	**54.2± 0.5**	33.7 ± 0.2	8
IR+SW	40.3 ± 0.5	26.7 ± 0.5	1
LeafJC	53.1 ± 0.5	33.7 ± 0.8	1.5
TreeJC	53.7 ± 0.6	**34.5± 0.3**	1.5

Table 1: Results on D2DStruct test set.

Model	F1	Exact Match	Train hrs
SW	52.4 ± 0.1	41.8 ± 0.3	32
Longformer	51.8 ± 0.2	41.4 ± 0.3	46
IR+SW	46.9 ± 0.4	35.6 ± 0.4	9
LeafJC	53.0 ± 0.4	42.6 ± 0.2	15
TreeJC	**54.9± 0.4**	**44.2± 0.3**	16

Table 2: Results on NQStruct test set.

4.1 Baselines

Sliding Window (SW) is a popular question answering baseline that trains a span selection model with Transformer encoding document trunks as described in Section 2.

Longformer is a Transformer model that handles long documents (Beltagy et al., 2020). We experiment the sliding window approach above with `Longformer-base` pretrained model with max sequence length of 4096 and a stride of 3072.

IR+SW is a pipeline approach that first identifies small number of k candidate paragraphs ($k = 10$ in the experiments here) via an information retrieval mechanism BM25 (Robertson et al., 1995), and then uses the SW approach. We consider it as a solution with reduced time complexity from the traditional SW approach for us to compare with.

LeafJC For ablation study, we experiment with a variant of TreeJC approach that excludes ancestors during encoding. The other implementation and experimental details are similar to TreeJC.

4.2 Experiment Settings

All models are implemented in PyTorch. Pretrained models are `Roberta-base` for SW, IR+SW, LeafJC and TreeJC, and `Longformer-base` for Longformer. Implementations of SW, Longformer and IR+SW are adapted from the SQuAD example code[1] in HuggingFace Transformers (Wolf et al.,

[1]`https://github.com/huggingface/transformers/blob/master/examples/legacy/question-answering/`. It is customized to fit the data format and to fix a tokenization issue in the original HuggingFace code.

Doc length (# tokens)	SW	TreeJC
≤ 0.5k	50.3/64.5	55.0/68.1
0.5k-1k	45.1/57.6	45.4/57.9
1k-2k	45.0/56.3	45.8/57.5
2k-4k	44.4/54.4	45.9/56.1
4k-8k	39.5/48.5	40.6/49.7
>8k	34.9/45.1	38.1/48.4

Table 3: Breakdown of results in `Exact_Match/F1` on NQStruct test set.

2019). For a fair comparison, input to SW, Longformer and IR+SW is flattened equivalent of the tree input to LeafJC and TreeJC and does not include the HTML tags of web pages.

All experiments were done on a single V100 GPU. In order to encode long sequences, Longformer requires much larger GPU memory, only 2 instances could fit in one V100 GPU, whereas 27 instances could fit in one V100 GPU with Roberta. Please see Appendix B for more details about experiment setup.

For the training of our approach TreeJC, positive instances are up-sampled to reach a balanced proportion of positive and negative training instances. To avoid the consequent bias towards longer documents, the loss from each example (document QA pair) is scaled down by the number of training instances from this example. λ and γ in Section 2 is set to be 0.5 and 1, respectively.

4.3 Results

For evaluation, we use exact match score and token-level F1 score (Rajpurkar et al., 2018). Table 1 and Table 2 present the evaluation results on the test sets of D2DStruct and NQStruct respectively along with the training time. All numbers are in the form of `mean` $\pm$ `std`, which is from three runs with different random seeds.

We observe consistent performance gains by TreeJC over almost all baselines. TreeJC shows a significant improvement over SW, which indicates the effectiveness of encoding the structure information with our joint model with cascaded inference. LeafJC performs better than SW but worse than TreeJC, which confirms the importance of including ancestor nodes during encoding. Longformer [2] serves as a competitive baseline and it achieves half a point higher F1 for D2DStruct dataset, however, at the cost of much longer training time. IR+SW method, on the other hand, shows high efficiency but suffers lower effectiveness, attributing to the fact that the IR method only achieves around 73% recall. In order to further examine how our approach performs on documents with different sizes, we break down the results on NQStruct dataset and compare the performances in Table 3. The results show that our approach has a clear gain on all document lengths over SW, especially on very long documents.

[2] 5 epochs were finished on NQStruct, as in experiments in (Beltagy et al., 2020).

5 Conclusion

We introduce a new Transformer-based method with joint learning and cascaded inference inspired by the tree structures of documents for machine reading comprehension. It outperforms several competitive baselines on two datasets from multiple domains. In particular, our study demonstrates that the proposed model is effective to encode longer documents with deep contexts for MRC tasks.

References

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284, Online. Association for Computational Linguistics.

Chris Alberti, Kenton Lee, and Michael Collins. 2019. A BERT baseline for the natural questions. *CoRR*, abs/1901.08634.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. 2017. Coarse-to-fine question answering for long documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 209–220, Vancouver, Canada. Association for Computational Linguistics.

Bhuwan Dhingra, Manzil Zaheer, Vidhisha Balachandran, Graham Neubig, Ruslan Salakhutdinov, and William W. Cohen. 2020. Differentiable reasoning over a virtual knowledge base.

Song Feng, Hui Wan, Chulaka Gunasekara, Siva Patel, Sachindra Joshi, and Luis Lastras. 2020. doc2dial: A goal-oriented document-grounded dialogue dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language*

Processing (EMNLP), pages 8118–8128, Online. Association for Computational Linguistics.

Mor Geva and Jonathan Berant. 2018. Learning to search in long documents using document structure. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 161–176, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Mandy Guo, Yinfei Yang, Keith Stevens, Daniel Cer, Heming Ge, Yun-hsuan Sung, Brian Strope, and Ray Kurzweil. 2019. Hierarchical document encoder for parallel corpus mining. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 64–72, Florence, Italy. Association for Computational Linguistics.

Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. 2019. Retrieve, read, rerank: Towards end-to-end multi-document reading comprehension. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2285–2295, Florence, Italy. Association for Computational Linguistics.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *International Conference on Learning Representations*.

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. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Jinhyuk Lee, Seongjun Yun, Hyunjae Kim, Miyoung Ko, and Jaewoo Kang. 2018. Ranking paragraphs for improving answer recall in open-domain question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 565–569, Brussels, Belgium. Association for Computational Linguistics.

Jiwei Li, Thang Luong, and Dan Jurafsky. 2015. A hierarchical neural autoencoder for paragraphs and documents. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1106–1115, Beijing, China. Association for Computational Linguistics.

Dayiheng Liu, Yeyun Gong, Jie Fu, Yu Yan, Jiusheng Chen, Daxin Jiang, Jiancheng Lv, and Nan Duan. 2020. RikiNet: Reading Wikipedia pages for natural question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6762–6771, Online. Association for Computational Linguistics.

Shanshan Liu, Xin Zhang, Sheng Zhang, Hui Wang, and Weiming Zhang. 2019. Neural machine reading comprehension: Methods and trends. *Applied Sciences*, 9(18):3698.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. 2013. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Marzieh Saeidi, Max Bartolo, Patrick Lewis, Sameer Singh, Tim Rocktäschel, Mike Sheldon, Guillaume Bouchard, and Sebastian Riedel. 2018. Interpretation of natural language rules in conversational machine reading. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2087–2097, Brussels, Belgium. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Tao Shen, Yi Mao, Pengcheng He, Guodong Long, Adam Trischler, and Weizhu Chen. 2020. Exploiting structured knowledge in text via graph-guided representation learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8980–8994, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.

Shuohang Wang, Mo Yu, Jing Jiang, Wei Zhang, Xiaoxiao Guo, Shiyu Chang, Zhiguo Wang, Tim Klinger, Gerald Tesauro, and Murray Campbell. 2018. Evidence aggregation for answer re-ranking in open-domain question answering. In *International Conference on Learning Representations*.

Shuohang Wang, Luowei Zhou, Zhe Gan, Yen-Chun Chen, Yuwei Fang, Siqi Sun, Yu Cheng, and Jingjing Liu. 2020. Cluster-former: Clustering-based sparse transformer for long-range dependency encoding.

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage BERT: A globally normalized BERT model for open-domain question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5878–5882, Hong Kong, China. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Ming Yan, Jiangnan Xia, Chen Wu, Bin Bi, Zhongzhou Zhao, Ji Zhang, Luo Si, Rui Wang, Wei Wang, and Haiqing Chen. 2019. A deep cascade model for multi-document reading comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7354–7361.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 5754–5764.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

Xingxing Zhang, Furu Wei, and Ming Zhou. 2019. HIBERT: Document level pre-training of hierarchical bidirectional transformers for document summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5059–5069, Florence, Italy. Association for Computational Linguistics.

Bo Zheng, Haoyang Wen, Yaobo Liang, Nan Duan, Wanxiang Che, Daxin Jiang, Ming Zhou, and Ting Liu. 2020. Document modeling with graph attention networks for multi-grained machine reading comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6708–6718, Online. Association for Computational Linguistics.

A Data

A.1 Data

NQStruct We derive a subset of 46k examples from about 100k training examples with short answers. From the set of 46K, we set aside a subset of 2.5K examples as our dev set. Similarly, we derive a subset of 2.3K examples from the original NQ dev set, and use that as our test set. In the cases that multiple short answers are available, we use the first one for training and evaluation. The number of examples in training/dev/test split is shown in Table 4. The distribution of document length is shown in Table 5.

D2DStruct The public release of Doc2Dial only provides train and dev sets [3]. For filtering out the non-answer agent turns, we filter out the cases where agent turn is grounded on the (sub)section titles. We combine the two and then create train/dev/test splits as 70%, 15%, 15% where 50% of the dev/test set are from documents unseen in training set. The number of examples in training/dev/test split is shown in Table 4. The distribution of document length is shown in Table 5.

# examples	**Training**	**Dev**	**Test**
NQStruct	43294	2500	2333
D2DStruct	7184	1065	1144

Table 4: Number of examples in datasets.

Dataset	**$\leq$ 0.5k**	**0.5k-1k**	**1k-2k**	**>2k**
NQStruct	7%	9%	15%	69%
D2DStruct	25%	45%	24%	6%

Table 5: Statistics of document length (in tokens).

B Experiment Settings

The deep learning systems are in PyTorch, and use Transformer encoder from HuggingFace Transformers. We use `Roberta-base` pretrained model and max sequence length of 512 unless otherwise stated. All experiments were done with fp16, on a single V100 GPU. Table 8 presents generation configurations and hyper-parameters that are shared by both datasets. For evaluation, we use the evaluation script 2.0 of SQuAD.
Table 9 presents configuration specifics for D2DStruct. For each dialogue in Doc2Dial, we combine all previous turns in reverse order as a query. Table 10 presents configuration specifics for NQStruct.

[3] http://doc2dial.github.io/data.html

C Results on Dev Sets

Table 6 and 7 present evaluation results on the dev sets of D2DStruct and NQStruct datasets respectively.

Model	**F1**	**Exact Match**
SW	53.0 $\pm$ 0.4	36.3 $\pm$ 0.4
Longformer	56.6 $\pm$ 0.8	37.0 $\pm$ 0.6
IR+SW	44.5 $\pm$ 0.4	31.3 $\pm$ 0.5
LeafJC	56.2 $\pm$ 0.3	39.7 $\pm$ 0.3
TreeJC	**57.3$\pm$ 0.4**	**41.0$\pm$ 0.3**

Table 6: Results on D2DStruct dev set.

Model	**F1**	**Exact Match**
SW	59.6 $\pm$ 0.3	49.3 $\pm$ 0.2
Longformer	57.8 $\pm$ 0.5	47.3 $\pm$ 0.5
IR+SW	50.6 $\pm$ 0.3	41.7 $\pm$ 0.4
LeafJC	58.5 $\pm$ 0.4	48.0 $\pm$ 0.5
TreeJC	**61.3$\pm$ 0.5**	**50.6$\pm$ 0.7**

Table 7: Results on NQStruct dev set.

	SW	**Longformer**	**IR+SW**	**LeafJC**	**TreeJC**
Total batch size	54	64	54	54	54
Batch size per GPU	27	2	27	27	27
Learning rate	3e-5	3e-5	3e-5	3e-5	3e-5
Weight decay	0.01	0.01	0.01	0.01	0.01
N best size	20	20	20	5	5
Warmup proportion	10%	10%	10%	10%	10%
Checkpoint freq	1/4 epoch	1/4 epoch	1/4 epoch	1/10 epoch	1/10 epoch

Table 8: General parameters

	SW	**Longformer**	**IR+SW**	**LeafJC**	**TreeJC**
# epochs	8	8	8	1	1
# instances per epoch	43, 961	7, 247	16, 441	266, 853	266, 951
max sequence length	512	4096	512	512	512
Document stride	128	3072	128	128	128
max query length	128	128	128	128	128
max answer length	30	30	30	30	30

Table 9: Dataset-specific parameters for D2DStruct experiments

	SW	**Longformer**	**IR+SW**	**LeafJC**	**TreeJC**
# epochs	8	5	8	1	1
# instances per epoch	879, 249	84, 788	275, 261	3, 232, 359	3, 241, 152
max sequence length	512	4096	512	512	512
Document stride	256	3072	256	256	256
max query length	64	64	64	64	64
max answer length	30	30	30	30	30

Table 10: Dataset-specific parameters for NQStruct experiments